공학박사학위논문

조합 구성적 FTL 설계 프레임워크에서의 쓰레기 수집 작업 정확성 검증

Correctness Verification of Garbage Collection Based on a Compositional FTL Design Framework

2014년 8월

서울대학교 대학원

전기·컴퓨터공학부

이 수 관

조합 구성적 FTL 설계 프레임워크에서의 쓰레기 수집 작업 정확성 검증

Correctness Verification of Garbage Collection Based on a Compositional FTL Design Framework

지도교수 조 유 근

이 논문을 공학박사 학위논문으로 제출함

2014년 7월

서울대학교 대학원

전기·컴퓨터공학부

이 수 관

이수관의 공학박사 학위논문을 인준함

2014년 7월

위 원 장 _____ (인)

부 위 원 장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

초 록

플래시 메모리는 내장형 시스템부터 고성능 서버 시스템의 저장 장치로까지 그 적용 범위가 점점 넓어지고 있다. 덮어쓰기를 허용하지 않는 플래시 메모리의 제약 조건을 극복하기 위한 다양한 연구를 통해 성능 면에서는 많은 발전이 이루어져 왔지만, 상대적으로 신뢰성 향상을 위한 연구는 많이 수행되지 않았다. 특히 최근에는 미세 공정의 도입 및 MLC(Multi-Level Cell) 플래시 메모리의 등장으로 신뢰성 특성이 점점 악화되고 있기 때문에 이를 매체로 사용하는 플래시 저장 장치 시스템의 신뢰성을 높이기 위한 연구가 더욱 중요해지고 있다.

저장 장치 시스템은 사용자 데이터 및 메타데이터에 대한 휘발성 상태와 비휘발성 상태를 모두 관리해야 되므로 매우 복잡한 구조를 갖는다. 이러한 저장 장치 시스템에서 비동기적으로 발생하는 전원 오류(power failure)로부터 정확한 일관성 있는 상태로 복구하는 것은 매우 어려운 작업이다.

본 논문은 체계적이고 완전한 전원 오류 복구를 지원하는 FTL 설계의 프레임워크인 HIL(Hierarchically Interacting set of Logs)을 기반으로 한다. HIL 프레임워크는 덮어쓰기를 허용하지 않는 플래시 메모리의 제약 조건을 고려하여 로그를 기본으로 하며, 호스트 데이터 및 FTL 메타데이터의 저장 및 관리를 로그와 각각의 로그 간의 상호 작용으로 처리한다.

본 논문은 FTL의 중요 기능 중 하나인 쓰레기 수집 작업에 초점을 맞추어 비동기적인 전원 오류가 발생하는 경우에도 HIL 프레임워크에 의해 설계

된 FTL은 정확한 저장 장치의 상태로 복구가 가능함을 검증한다. 이를 위하여 쓰레기 수집 작업을 위한 물리 블록 정보 로그를 정의하고 사용자 데이터 및 매핑 관련 FTL 메타데이터를 저장하기 위한 로그들과의 전원 오류 복구를 위한 인터페이스 및 로그 운영 규칙을 제시한다. 이를 통해 HIL 프레임워크를 이용해 설계된 FTL이 비동기적인 전원 오류 발생 시에도 정확한 저장 장치 상태로 복구될 수 있음을 검증한다. 또한 쓰레기 수집 작업 동안 불필요한 논리 주소 중복 검사로 인해 발생할 수 있는 성능 저하를 최소화하기 위하여 가상 클록 및 블룸 필터를 활용하는 기법을 제시한다.

주요어: 전원 오류 복구, 플래시 메모리, 플래시 메모리 기반 저장 장치, 플래시 변환 계층(FTL), 쓰레기 수집

학 번: 2002-21581

목 차

제 1 장 서 론 ····· 1

1.1 연구 동기 ····· 1
1.2 연구 내용 ····· 6
1.3 논문의 구성 ····· 7

제 2 장 배경 지식 및 관련 연구 ····· 9

2.1 플래시 메모리의 구조 및 특성 ····· 9
2.2 플래시 메모리 소프트웨어 ····· 14
2.3 쓰레기 수집 작업 ····· 20
2.4 전원 오류 복구 관련 문제점 및 기술 추세 ····· 24

제 3 장 HIL 프레임워크 및 쓰레기 수집 작업 ····· 27

3.1 FTL 추상화 ····· 27
3.2 HIL 프레임워크 설계 ····· 30
3.2.1 HIL 프레임워크 구성 요소: 로그 ····· 30
3.2.2 로그 간의 계층 구조와 상호 작용 ····· 33
3.2.3 로그 조합을 통한 FTL 구성 ····· 35
3.3 HIL 프레임워크 동작 ····· 36
3.3.1 호스트 쓰기 요청 처리 ····· 36
3.3.2 호스트 읽기 요청 처리 ····· 39

3.3.3 쓰레기 수집 동작 절차 ····· 40
3.4 HIL 프레임워크에서의 전원 오류 복구 ····· 42
3.4.1 구조적 복구 (structural recovery) ····· 42
3.4.2 기능적 복구 (functional recovery) ····· 44

제 4 장 쓰레기 수집 동작의 정확성 검증 ····· 48

4.1 쓰레기 수집 동작시 전원 오류 복구 문제와 회피 규칙 ····· 49
4.1.1 형제 페이지에 의한 소거 오류 문제 ····· 49
4.1.2 병렬성에 의한 유효 페이지 복사 문제 ····· 52
4.2 쓰레기 수집 동작 정확성 검증 ····· 55
4.2.1 정확성 기준(correctness criteria) ····· 55
4.2.2 쓰레기 수집 작업의 정확성 검증 ····· 56

제 5 장 쓰레기 수집 성능 최적화 ····· 61

5.1 블룸 필터(Bloom Filter) ····· 61
5.2 가상 클록과 블룸 필터를 이용한 쓰레기 수집 성능 향상 ····· 63

제 6 장 결론 및 향후 연구 과제 ····· 68

참고문헌 ····· 71

Abstract ····· 75

그림 목차

그림 1 NAND 플래시 메모리 셀 구조 및 동작 ····· 10
그림 2 NAND 플래시 메모리 칩 내부 구조 ····· 11
그림 3 SLC 플래시 메모리 셀과 MLC 플래시 메모리 셀 ····· 13
그림 4 플래시 변환 계층의 역할 ····· 14
그림 5 호스트 쓰기 처리에 따른 페이지 무효화 과정 ····· 21
그림 6 페이지 매핑 FTL에서의 쓰레기 수집 작업 ····· 22
그림 7 블록 매핑 FTL에서의 쓰레기 수집 작업 ····· 23
그림 8 HIL 프레임워크에서의 로그 동작 및 내부 구조 ····· 30
그림 9 매핑 로그의 동작 및 내부 구조 ····· 32
그림 10 물리 블록 정보 로그의 동작 및 내부 구조 ····· 33
그림 11 로그 들 간의 계층 구조 ····· 34
그림 12 로그 조합을 통한 다양한 FTL 구성 ····· 35
그림 13 HIL 프레임워크에서의 호스트 쓰기 요청 처리 과정 ····· 37
그림 14 HIL 프레임워크에서의 호스트 읽기 요청 처리 과정 ····· 39
그림 15 HIL 프레임워크에서의 쓰레기 수집 동작 절차 ····· 41
그림 16 프로그램 중 전원 오류 발생에 따른 페이지 상태 ····· 42
그림 17 무해석 블록을 이용한 잔여 효과 제거 ····· 44
그림 18 전원 오류 발생 시 최신 정보의 소실 예 ····· 45
그림 19 기능적 복구를 위한 자료 구조 ····· 46
그림 20 형제 페이지에 의한 소거 오류 문제 ····· 50

그림 21 병렬성에 의한 유효 페이지 복사 문제 ····· 53
그림 22 쓰레기 수집 작업의 정확성 검증 세부 단계 ····· 57
그림 23 로그들 간의 유효성 관점의 차이 ····· 58
그림 24 블룸 필터로의 원소 삽입과 질의 ····· 62
그림 25 시간 관점에서 본 유효 페이지 복사 과정 시 동기화 문제 ····· 63
그림 26 다수 개의 블룸 필터를 이용한 방법 ····· 66

제 1 장 서 론

1.1 연구 동기

컴퓨터 시스템의 주된 용도 중 하나는 데이터를 안정적으로 저장하고 효율적으로 추출하는 것이다. 이러한 작업을 담당하는 저장 장치 시스템은 파일 시스템, 페이지 캐시, 블록 장치 계층 등의 운영 체제 소프트웨어부터 물리적인 저장 장치 내부의 펌웨어 및 하드웨어까지 모두 연관되는 매우 복잡한 구조로 이루어진다. 이러한 저장 장치 시스템의 설계에 있어 중요한 두 축은 성능(performance) 및 신뢰성(reliability)이라고 할 수 있다.

현재까지 컴퓨터 시스템에서 비휘발성 저장 매체로 가장 많이 사용되고 있는 것은 하드 디스크이며 이에 따라 저장 장치 시스템은 하드 디스크의 특성에 최적화되어 설계되어져 왔다. 성능 측면에서는, 응답시간과 직접적인 관련이 있는 하드 디스크의 기계적인 헤드 움직임을 최소화하기 위한 Berkeley Unix Fast File System(Unix FFS)[1]을 기반으로 하는 전통적인 파일 시스템 외에도 로그 구조 파일 시스템(Log-structured file system)[2] 등의 다양한 파일 시스템이 개발 되어 왔다. 신뢰성 측면에서는, 비동기적으로 발생하는 전원 오류(power failure)로부터 저장 장치 시스템의 신뢰성을 확보하기 위한 방법으로 저널링에 기반한 다양한 기법들이 제안되어져 왔다[3].

플래시 메모리는 EEPROM의 일종인 비휘발성 메모리로서 전력 소모가 적고 크기가 작으며 충격 및 진동에 강한 특성을 지니기 때문에 초기에는 내장형 시스템(embedded system)이나 이동형 장치(mobile device)의 저장 매체로 많

이 사용되어져 왔다. 최근에는 NOR 플래시 메모리 대비 월등히 높은 집적도를 갖는 NAND 플래시 메모리와 하나의 메모리 셀(memory cell)로 복수 개의 비트(bit)를 표현하는 MLC(Multi-Level Cell) 플래시 메모리가 등장하였으며, 제조 공정의 발달 및 수요의 급격한 증대로 인해 가격 대비 용량이 급속하게 증가하고 있다. 이에 따라 SSD(Solid State Drive)의 형태로 노트북이나 개인용 컴퓨터의 하드 디스크를 대체하거나 고성능 서버 시스템의 저장 장치로까지 사용되는 등 그 적용 범위가 점점 넓어지고 있다.

동작 방식 면에서 살펴보면, 섹터 단위의 읽기 및 쓰기를 기본 동작으로 하는 하드 디스크와 달리, 플래시 메모리는 페이지 단위의 읽기 및 쓰기를 기본 동작으로 하며 덮어쓰기(overwrite, in-place update)를 허용하지 않는다. 하나의 플래시 메모리 칩은 다수 개의 블록(block) 들로 구성되며, 각각의 블록은 다시 다수 개의 페이지(page)로 이루어진다. 한번 쓰기 동작이 가해진 페이지에는 덮어쓰기가 허용되지 않으며, 새로운 데이터를 쓰기 위해서는 해당 페이지가 속해 있는 블록 전체에 대해 지우기(소거) 작업이 선행되어져야 한다.

성능(performance) 관점에서 볼 때, 기계적인 동작이 필요하여 해당 데이터에 대한 접근 시간(seek time)이 매우 오래 걸리는 하드 디스크에 비해, 플래시 메모리는 순수하게 전기적으로 동작하는 소자이기 때문에 임의 접근 시간이 월등히 빠르다는 장점이 있다. 하지만 전술한 바와 같이 플래시 메모리는 단위 페이지에 대한 쓰기 요청이 한번 수행되면 해당 페이지에 새로운 데이터를 덮어쓸 수 없고, 해당 페이지가 속해 있는 블록 전체에 대해 소거 작업을 수행해야 하기 때문에 부가적인 지우기 및 복사 연산이 필요하며 이는 플래시 메모리 저장 장치의 성능 저하를 가져오게 되는 원인이 된다.

신뢰성(reliability) 관점에서 볼 때, 플래시 메모리는 다수의 좋지 않은 특성을 가지고 있다. 일반적으로 플래시 메모리는 수율을 높이기 위해 공장 출하 시부터 정상적인 동작을 보장할 수 없는 메모리 셀이 포함된 소량의 불량 블록의 존재를 허용하며, 사용 시간이 길어짐에 따라 메모리 셀의 열화에 의해 정상적인 블록도 불량 블록이 될 수 있다. 이러한 플래시 메모리 셀의 내구성은 블록이 허용하는 쓰기 및 소거 횟수(program and erase cycle)로 표현된다. 초기에 개발된 SLC(Single Level Cell) 타입의 NAND 플래시 메모리는 대략 100,000번 정도를 허용하였으며, 최근에 많이 사용되는 MLC(Multi Level Cell) 타입의 NAND 플래시 메모리는 SLC 타입에 비해 훨씬 적은 횟수만을 허용한다. 또한 메모리 셀에 대한 읽기/쓰기 작업 처리 시, 주변에 위치하는 다른 메모리 셀 들이 간섭을 받아 저장하고 있는 값이 변하게 되는 읽기/쓰기 교란(read/write disturbance)의 확률도 현저하게 증가하고 있다

위와 같은 다양한 제약 조건 및 성능 및 신뢰성 특성으로 인해, 플래시 메모리 자체만으로는 컴퓨터 시스템의 파일 시스템 및 가상 메모리의 비휘발성 대용량 저장 매체로 하드 디스크를 직접 대체할 수 없으며, 일반적으로 FTL(Flash Translation Layer)이라고 불리는 소프트웨어가 운영 체제의 일부 또는 저장 장치 내부의 펌웨어 형태로 구현된다. FTL은 호스트 시스템의 논리 주소 공간에 대한 섹터 단위의 쓰기 및 읽기 요청을 내부 매핑(mapping)을 통하여 플래시 메모리의 물리 주소 공간에 대한 페이지 단위의 쓰기 및 읽기 요청으로 변환하여 처리한다. 제자리 갱신을 허용하지 않는 제약 조건을 회피하기 위하여 호스트 쓰기 요청은 FTL이 관리하는 미리 소거된 영역에 쓰여지고 내부적으로 유지 및 관리하는 매핑 정보를 갱신하며, 호스트 읽기 요청 역시 매핑테이블을 통해

물리 주소를 알아낸 후 해당 데이터를 읽어 호스트로 전달한다.

플래시 메모리 기반 저장 장치가 사용됨에 따라 무효화된 페이지는 점점 증가하고 사용 가능한 소거 영역은 점점 줄어들게 된다. 이에 따라, FTL은 무효화된 페이지들로 이루어진 블록들을 선정하고 소거 작업을 수행하여 쓰기 가능 블록들을 만들어내야 하며, 이러한 작업을 쓰레기 수집(garbage collection) 또는 블록 병합(block merge) 작업이라고 부른다. 이는 로그 구조 파일 시스템에서의 세그먼트 클리닝(segment cleaning)과 유사한 개념이며, 호스트 요청과 무관하게 FTL 내부의 필요에 의해 발생하는 작업으로 저장 장치 성능을 저하시키는 큰 요인이 되기 때문에 이를 최소화시키기 위한 많은 연구가 수행되었다[4][5][6][7][8].

FTL은 저장 장치 시스템의 높은 신뢰성 요구 수준을 만족시키기 위해 플래시 메모리의 다양한 신뢰성 특성에 대한 고려를 해야 한다. 초기 불량 블록(initial bad block) 및 동작 중 발생 가능한 불량 블록(runtime bad block) 문제를 처리하기 위해, 플래시 메모리의 일부 영역을 비축(reserve)해 놓고, 불량 블록 발생 시 교체하는 불량 블록 관리 기법이 도입되었으며[9][10], 특정 블록에 소거 및 쓰기 요청이 집중될 경우 신뢰성이 저하되는 문제를 해결하기 위해 플래시 메모리 전체 블록에 대해 소거 및 쓰기 요청을 고르게 분산시키기 위한 마모도 평준화 (wear leveling) 기법이 사용된다[11].

이러한 FTL의 구성 및 알고리즘은 플래시 메모리 기반 저장 장치 시스템의 성능에 직접적인 영향을 주기 때문에 현재까지 다양한 형태의 FTL이 제안되고 구현되었다. 하지만 이러한 연구 들은 주로 성능 향상의 관점에서 수행되어져

왔고 저장 장치 시스템 설계의 중요한 한 축인 신뢰성 측면에서는 상대적으로 많은 연구가 이루어지지 않았다.

최근 플래시 메모리의 개발 동향을 살펴보면, 공정의 미세화 및 MLC(Multi Level Cell) 타입 플래시 메모리의 등장으로 인해 신뢰성 특성이 급격하게 악화되고 있다. MLC 플래시 메모리는 하나의 메모리 셀로 2비트 또는 3비트의 상태를 표현하는 플래시 메모리로, 하나의 메모리 셀로 1비트를 표현하는 기존의 SLC(Single Level Cell) 대비 용량 면에서는 우수하나 신뢰성 특성이 좋지 않다. MLC 플래시 메모리는 허용하는 쓰기 및 소거 횟수의 제한이 SLC 대비 1/10 수준으로 감소하였으며, 하나의 메모리 셀에 대한 읽기 및 쓰기 처리 시, 주변에 위치하는 다른 메모리 셀 들이 간섭을 받아 저장하고 있는 값이 변하게 되는 읽기 교란(read disturbance)과 쓰기 교란(write disturbance)의 확률이 현저하게 증가하고 있다[12]. 또한 MLC 구조에서는 하나의 메모리 셀이 표현하는 복수 개의 비트가 동일 블록의 상이한 페이지에 분산되어 있기 때문에, 어떤 과거의 한 시점에서 쓰기 동작이 정상적으로 완료되었다 하더라도 추후에 짝을 이루는 다른 페이지에 대한 쓰기 동작 중 전원 오류가 발생하면 기존의 페이지에 기록된 데이터도 손상을 받게 된다. 이렇게 내부 구조에 의해 쌍으로 연결이 되어 있는 페이지 들을 형제 관계 페이지(sibling page)라고 부른다. 이러한 제약 조건은 기존의 SLC 플래시 메모리에는 존재하지 않았던 것이므로, 기존의 SLC 플래시 메모리를 위해 개발된 FTL을 MLC 플래시 메모리에 단순 적용할 경우 저장 장치 시스템의 정확성을 보장받을 수 없게 된다.

높은 성능 및 용량에 대한 요구가 크지 않았던 과거의 플래시 메모리 기반 저장 장치 시스템에서는, 체계적이지 않은 단순한 기법으로도 요구되는 최소한의

신뢰성을 비교적 쉽게 얻을 수 있었다. 하지만 최근에 플래시 메모리는 대형 시스템의 고성능 저장 장치 시스템으로까지 적용 범위가 계속 넓어지고 있으며, 성능의 극대화를 위해 다수의 칩 및 다수의 채널을 사용해 인터리빙(interleaving) 및 병렬성(parallelism)의 수준을 극대화하고, 디바이스 내의 RAM을 이용한 캐싱 및 버퍼링을 적극적으로 이용하는 등 FTL의 복잡도는 급속하게 높아지고 있다. 이러한 환경에서 저장 장치 시스템으로서 요구되는 충분한 신뢰성을 확보하는 것은 점점 더 어려운 작업이 되고 있다.

1.2 연구 내용

본 연구는 다양한 매핑 알고리즘으로 대표되는 FTL 구현에 독립적으로, 전원 오류 시 발생 가능한 문제점을 해결하고 일관성 있는 상태로 저장 장치 시스템의 복구를 가능케 하는 전원 오류 복구 프레임워크인 HIL(Hierarchically Interacting set of Logs)을 기반으로 한다. FTL의 동작은 기본적으로 매핑을 이용한 호스트 시스템의 읽기 및 쓰기 처리와, 소거 작업을 통해 쓰기 가능 블록들을 만들어 내기 위한 쓰레기 수거 작업으로 나누어 질 수 있다. 호스트 시스템의 읽기 및 쓰기 처리에 대한 HIL 프레임워크의 전원 오류 복구 기법의 정확성은 조합 구성적 FTL 설계 프레임워크[13]의 연구를 통해 정형적으로 증명된 바 있으며, 본 연구는 이를 확장하여 쓰레기 수거 작업이 수행되는 경우에도 HIL 프레임워크의 전원 오류 복구의 정확성은 여전히 유지된다는 것을 증명하는 것을 목표로 한다.

HIL 프레임워크의 핵심 구성 요소는 로그(log)이며, 사용자 데이터 및 FTL 메타데이터 등, FTL에 의해 휘발성 또는 비휘발성으로 관리되는 모든 데이터는

로그로 구성된다. 플래시 메모리는 제자리 갱신을 허용하지 않기 때문에 붙여 쓰기만 허용하는(append-only write) 로그는 플래시 메모리에 데이터를 기록하는 자연스러운 방법이라고 할 수 있다. 이러한 각각의 로그 들은 하나의 쓰레드(thread)가 모든 작업을 순차적으로 처리하는 것도 가능하지만, 성능 향상을 위해 다수의 플래시 메모리, 다수의 플래시 메모리 컨트롤러, 다수의 프로세서에 자유롭게 할당 되어 최대한의 병렬성을 제공하는 것을 기본 모델로 한다. HIL 프레임워크에서 사용되는 로그의 종류를 살펴보면 먼저 호스트 데이터를 저장하는 데이터 로그(data log)가 있고, 이러한 호스트 데이터의 논리 주소와 물리 주소의 연결 정보를 유지하는 매핑 로그(mapping log)가 있다. 또한 각각의 로그 들이 사용됨에 따라 변경되는 페이지의 유효성 정보 및 물리 블록과 로그 간의 할당 정보를 관리 및 저장하는 물리 블록 정보 로그(liveness log)가 있다. 이 외에 HIL 프레임워크 자체의 메타데이터라 할 수 있으며, 내부적으로 전원오류 복구에 필요한 각 로그에 대한 관리 정보를 담고 있는 체크포인트 로그(checkpoint log)가 있다. HIL 프레임워크는 이러한 기본적인 로그들의 존재를 가정하고 로그들 간의 계층 구조와 동작 규칙을 명시함으로써 완전하고 증명 가능한 전원 복구 프레임워크를 제공한다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다. 제 2장에서는 플래시 메모리에 대한 기본적인 배경 지식을 설명한 후 FTL 매핑 기법 및 신뢰성 확보 기법에 대한 관련 연구를 살펴본다. 제 3장에서는 FTL의 구현에 독립적인 전원 오류 복구 프레임워크인 HIL에 대하여 설명한다. 제 4장에서는 쓰레기 수집 작업과 관련하여 필요한 로그 타입과 로그 운영 규칙을 제시하고 이를 통해 HIL 프

레임워크가 쓰레기 수집 작업이 수행되는 경우에도 비동기적인 전원 오류 발생 시에 정확히 저장 장치의 일관된 상태로 복구 가능함을 보인다. 제 5장에서는 제시한 규칙을 준수함에 따라 발생할 수 있는 성능 저하를 최소화하기 위하여 가상 클록 및 블룸 필터를 활용하는 기법을 제시하였다. 마지막으로 제 5장에서는 연구 결과를 요약하고 향후 연구 방향에 대해 논의한다.

제 2 장　배경 지식 및 관련 연구

본 장에서는 플래시 메모리의 구조 및 특성을 기술하고, 플래시 메모리의 제약 조건을 감추어 일반적인 저장 장치 시스템의 저장 매체로 사용할 수 있도록 하는 소프트웨어인 FTL에 대하여 설명한다. 또한 FTL의 주요 기능인 호스트 쓰기 및 읽기 요청 처리에 대해 설명하고, 쓰기 가능한 블록을 만들기 위한 쓰레기 수집 작업과 관련된 연구 및 전원 오류 복구에 대해서 기술한다.

2.1 플래시 메모리의 구조 및 특성

플래시 메모리는 데이터를 전기적으로 지우고 재기록할 수 있는 EEPROM(Electrically Erasable Programmable Read-Only Memory)의 일종으로 기계적인 동작이 없는 순수 반도체로 이루어진 비휘발성 메모리(non-volatile memory)이다. 플래시 메모리는 전력 소모가 적고 크기가 작으며 충격 및 진동에 강한 특성을 지녀 주로 내장형 시스템(embedded system)이나 이동형 장치(mobile device)의 저장 매체로 사용되어져 왔다. 플래시 메모리는 내부 메모리 셀의 구성 형태에 따라 크게 NOR 플래시 메모리와 NAND 플래시 메모리로 구분된다. 초기에 개발된 NOR 플래시 메모리는 SRAM 형태의 인터페이스를 지원하여 바이트 단위의 읽기가 가능하기 때문에 프로그램 코드의 직접 실행(XIP: eXecute In Place) 용도로 많이 사용되어져 왔고, 그 후에 등장한 NAND 플래시 메모리는 NOR 플래시 메모리에 비해 집적도가 비약적으로 높고 쓰기 속도가 빠르기 때문에 주로 사용자 데이터 저장의 용도로 많이 사용되어져 왔다. 본 연구는 플래시 메모리 기반 저장 장치 매체로 주로 사용되는 NAND 플래시 메모리만을 대상으로 한다.

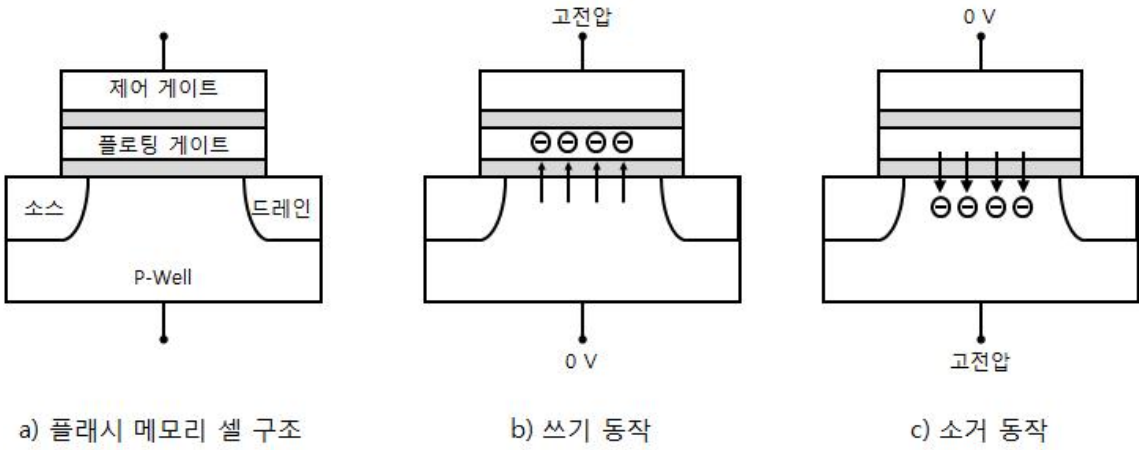

그림 1 NAND 플래시 메모리 셀 구조 및 동작

그림 1-(a)는 NAND 플래시 메모리를 구성하는 최소 단위인 메모리 셀의 구조를 보여준다. NAND 플래시 메모리 셀은 일반적인 MOSFET(Metal Oxide Semiconductor Field Effect Transistor) 트랜지스터와 유사하지만, 제어 게이트(control gate)와 기판(substrate) 사이에 산화막으로 절연된 플로팅 게이트(floating gate)가 추가된다. 그림 1-(b) 는 플래시 메모리 셀에 대한 쓰기 또는 프로그램(program) 동작을 보여준다. P-Well 에 0V 를 주고 제어 게이트에 높은 전압을 가하게 되면 전기장의 변화로 인해 전자들이 얇은 산화막을 통과하여 플로팅 게이트로 이동하게 되는데 이를 터널링 효과(tunneling effect)라고 부른다. 이렇게 플로팅 게이트에 쌓인 전자들은 제어 게이트에 인가된 전원이 차단된 후에도 플로팅 게이트에 갇혀 있게 되며 이를 논리적 '0'의 상태로 정의한다. 그림 1-(c)는 플래시 메모리 셀의 소거(erase) 동작을 보여준다. 제어 게이트에 0V 를 주고 P-Well에 높은 전압을 가하게 되면 플로팅 게이트에 갇혀 있던 전자가 빠져나오게 되며 이를 논리적 '1'의 상태로 정의한다. 플로팅 게이트

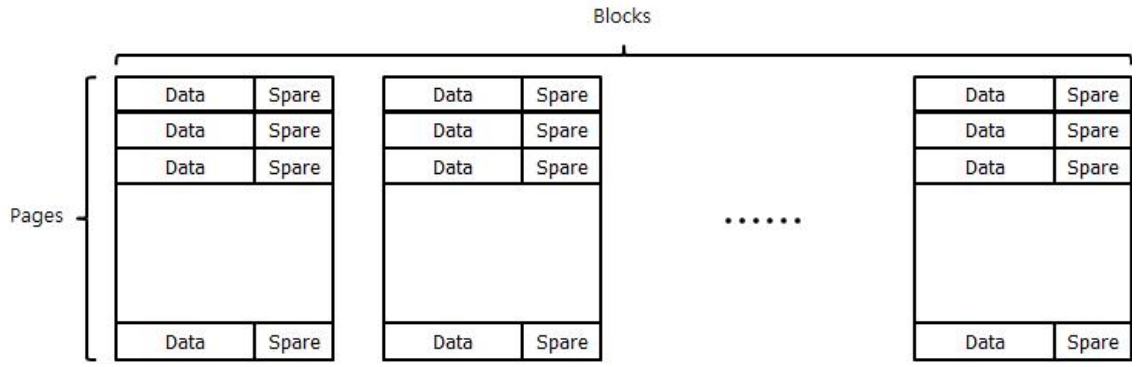

그림 2 NAND 플래시 메모리 칩 내부 구조

에 갇혀 있는 전자의 양에 따라 플래시 메모리 셀의 문턱 전압(threshold voltage)이 변경되며 이에 따라 소스에서 드레인으로 전압을 가해줄 경우 전류가 흐르게 될지의 여부가 결정된다. 이를 센싱하면 현재 플래시 메모리 셀의 논리적 상태를 판단할 수 있게 된다.

초기의 NAND 플래시 메모리는 하나의 메모리 셀로 단일 비트를 표현하는 SLC(Single Level Cell) 타입이었지만, 최근에는 하나의 메모리 셀로 복수 개의 비트를 표현하는 MLC(Multi Level Cell) 타입의 플래시 메모리가 많이 사용되고 있다. MLC 타입 플래시 메모리는 플로팅 게이트에 저장되는 전자의 양을 조절하여 다양한 문턱전압의 단계를 표현할 수 있다. 2 비트 MLC는 하나의 메모리 셀이 네 단계의 상태를 나타내며, 3 비트 MLC는 하나의 메모리 셀이 여덟 단계의 상태를 나타내게 된다. 이렇게 하나의 셀로 다수의 상태를 표현하기 때문에 표현 가능한 정보량이 증가하여 플래시 메모리의 저장 용량을 크게 증가시킬 수 있다는 장점이 있지만, 읽기 및 쓰기 동작 시 상태 정보를 확인하기 위하여 훨씬 세밀한 센싱이 필요하기 때문에 성능 및 신뢰성 특성이 저하되는 문제

가 있다.

그림 2는 일반적인 NAND 플래시 메모리 칩의 내부 구조를 보여준다. NAND 플래시 메모리 칩은 소거 동작의 단위가 되는 다수 개의 블록(block) 들로 구성되며, 각각의 블록은 다시 읽기 및 쓰기의 단위가 되는 다수 개의 페이지(page)로 구성된다. 페이지는 사용자 데이터를 저장하는 데이터 영역(data region) 및 부가 정보를 기록할 수 있는 소량의 예비 영역(spare region)으로 이루어진다. 예비 영역은 일반적으로 데이터 영역에 기록되는 데이터의 에러를 검출하고 복구하기 위한 ECC(Error Correcting Code) 정보 및 FTL이 사용하는 매핑 정보를 기록하는 용도로 사용된다. 데이터 영역의 크기는 일반적인 저장 장치 시스템에서 읽기, 쓰기의 단위로 사용되는 512B의 배수이며 최근에는 그 크기가 점점 커지고 있는 추세를 보이고 있다. 예비 영역의 크기도 데이터 영역의 크기 증가 및 플래시 메모리의 신뢰성 저하에 따른 더욱 강력한 ECC 알고리즘의 필요성에 의해 크기가 점점 증가하고 있다.

플래시 메모리는 신뢰성 측면에서 많은 단점을 지니고 있다. 플래시 메모리 셀은 수명이 영구적이지 않으며 허용하는 쓰기 및 소거의 횟수(program & erase cycle)에 제한이 있다. 일반적으로 SLC 타입 플래시 메모리의 경우 100,000번, MLC 타입 플래시 메모리의 경우는 그보다 훨씬 적은 횟수를 허용하며, 이보다 많은 쓰기 및 소거가 가해질 경우 해당 플래시 메모리 셀의 안정적인 동작을 보장할 수 없게 된다. 또한 정상적인 플래시 메모리 셀로만 이루어진 플래시 메모리 칩을 제조하는 것은 매우 어렵기 때문에 수율(yield)을 높이기 위해 소량의 초기 불량 블록을 허용한다. 그러므로 플래시 메모리를 처음 사용할 때 이러한 불량 블록을 검출하여 사용 가능 블록 리스트에서 배제하고, 동작 중

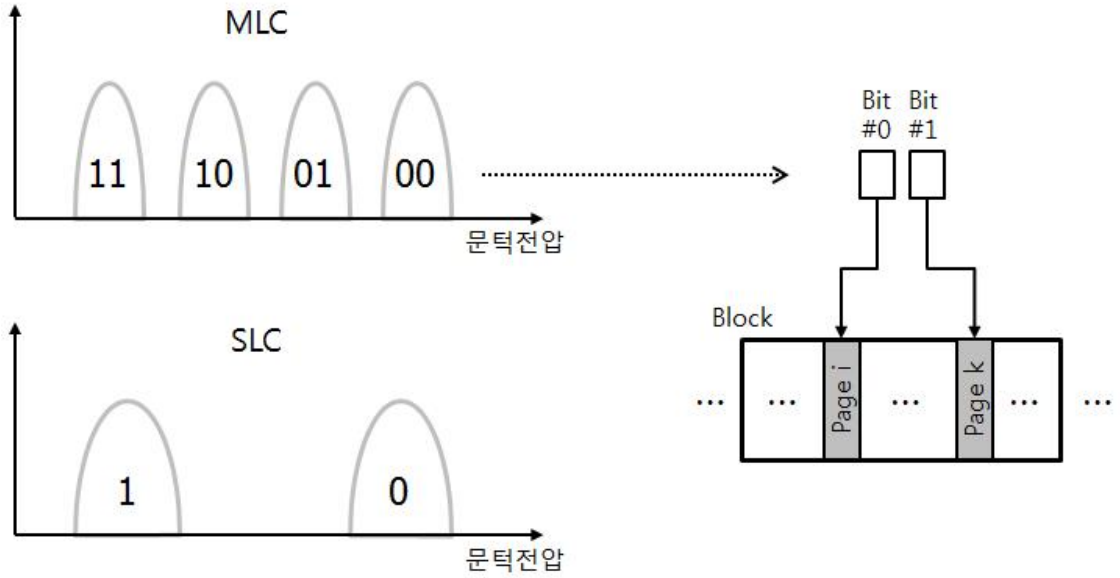

그림 3 SLC 플래시 메모리 셀과 MLC 플래시 메모리 셀

발생하는 불량 블록에 대해서는 일정량의 정상 블록을 비축하였다가 교체하여 주는 작업이 반드시 필요하다. 제조 공정이 점점 미세화 됨에 따라 플래시 메모리 셀의 신뢰성은 더욱 악화되고 있으며, 플로팅 게이트에 갇혀 있는 전자들이 새어 나옴에 따라(leakage) 발생하는 데이터 보유 문제(retention), 하나의 플래시 메모리 셀에 가해지는 쓰기 동작에 의해 인접한 셀들이 영향을 받아 데이터가 변하게 되는 쓰기 교란(write disturbance) 등 많은 문제가 발생하고 있다[12]. 그림 3은 MLC 플래시 메모리에 존재하는 형제 페이지(sibling page) 구조를 나타낸다. MLC 플래시 메모리 셀이 표현하는 두 비트는 같은 페이지가 아닌 상이한 두 페이지에 할당되며 이를 형제 페이지(sibling page) 관계라고 한다. 그림 3에서 보이는 형제 페이지 관계에서 LSB 페이지(Page i)는 MSB 페이지(Page k)에 대한 쓰기 동작이 완료된 후에만 영속성이 보장된다. 만약 MSB

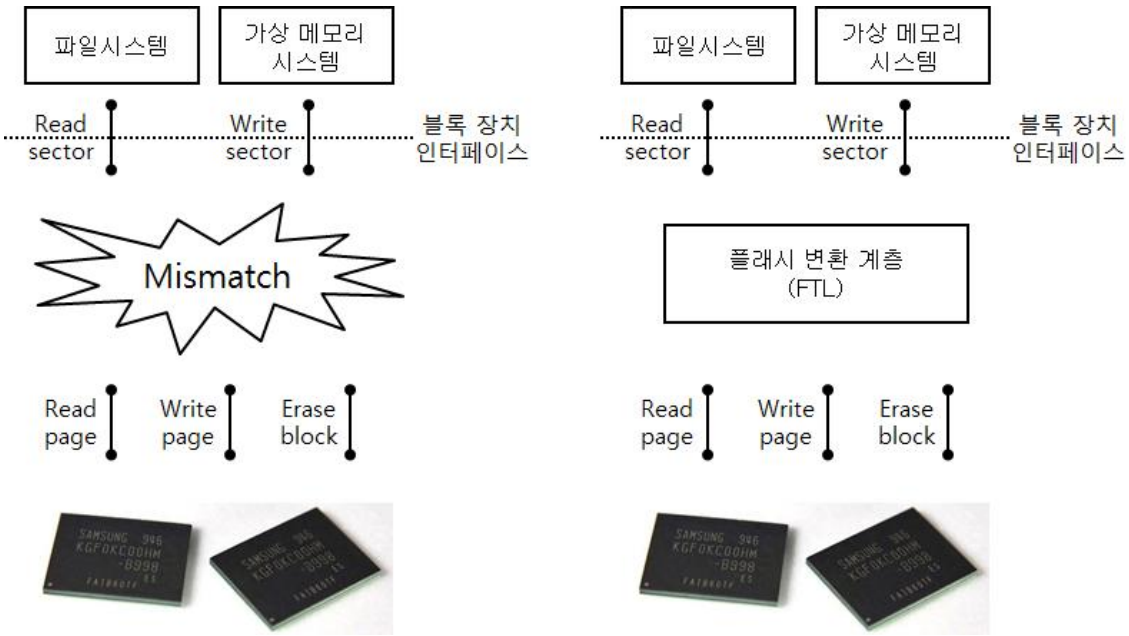

그림 4 플래시 변환 계층의 역할

페이지를 쓰는 도중에 전원 오류가 발생할 경우 LSB 페이지의 상태도 변하게 되는 문제가 발생한다.

2.2 플래시 메모리 소프트웨어

플래시 메모리는 전술한 것처럼 하드 디스크와 달리 덮어쓰기(overwrite, in-place update)가 불가능하다는 제약 조건이 있다. 또한 섹터 단위의 읽기 및 쓰기를 기본 동작으로 하는 하드 디스크와 달리 페이지 단위의 읽기 및 쓰기, 블록 단위의 지우기를 기본 동작으로 한다. 이러한 차이점 및 제약 조건으로 인해 플래시 메모리 자체만으로는 하드 디스크를 직접 대체하여 호스트 시스템의 저장 매체 역할을 수행할 수 없으므로 플래시 변환 계층(FTL; Flash Translation Layer)으로 불리는 소프트웨어가 필수적으로 사용된다.

FTL은 그림 4에서 보여지는 것처럼, 플래시 메모리의 고유한 특성 및 제약 조건을 숨기고 호스트 시스템에게 하드 디스크와 동일한 블록 장치 인터페이스를 제공하는 소프트웨어이며 이를 통해 호스트 시스템의 파일 시스템 또는 가상 메모리 시스템에 아무 수정 없이 플래시 메모리 기반 저장 장치가 사용될 수 있다.

FTL의 가장 중요한 기능은 블록 장치 인터페이스를 통해 호스트 시스템에게 보여주는(export) 섹터 기반의 논리 주소와 실제로 플래시 메모리에 기록되는 블록, 페이지 기반의 물리 주소의 매핑을 관리하는 것이다. 호스트 시스템에서 쓰기 요청을 받게 되면 FTL은 미리 지워둔 영역을 이용해 호스트 데이터를 기록한 후 내부적으로 유지하는 논리 주소와 물리 주소의 매핑 정보를 갱신한다. 이러한 매핑 정보는 일반적으로 테이블 형태를 갖게 되며 FTL 메타데이터로 불린다. 추후에 해당 논리 주소에 대한 읽기 요청이 전달되면 내부적으로 유지하는 매핑 정보 테이블을 이용하여 해당 데이터가 저장되어 있는 물리 주소를 찾을 수 있다. FTL은 또한 수율 증가를 위해 제조 시 허용하는 초기 불량 블록(initial bad block) 및 플래시 메모리 사용 시간이 증가함에 따라 발생하는 런타임 불량 블록(runtime bad block)에 대한 처리도 담당한다. 저장 장치 초기화 시에 FTL은 전체 블록을 검사하여 초기 불량 블록을 검출한 후 사용 가능 블록 리스트에서 제외시켜야 하며, 동작 중에 발생할 수 있는 런타임 불량 블록을 교체하기 위해 소량의 가용 블록을 예비 블록으로 비축(reserve)한다. 플래시 메모리는 블록의 소거 횟수에 제한이 있기 때문에 사용 시간이 길어짐에 따라 런타임 불량 블록이 발생할 수 있다. 이 때 특정 블록에 쓰기 동작이 집중이 되지 않고 전체 블록들의 소거 횟수가 균등하게 유지되도록 하는 마모도 평준화

(wear-leveling) 작업이 필요하다. 런타임 불량 블록이 발생하면 FTL은 비축해둔 예비 블록을 이용하여 교체 작업을 수행하게 된다.

FTL의 매핑 방식은 플래시 저장 장치의 성능에 직접적인 영향을 미치기 때문에 많은 연구 그룹에 의해 다양한 기법이 연구되고 제안되어져 왔다[4][5][6][7][18][19][20][21][22][23]. 이러한 기법은 크게 매핑 단위의 크기(mapping granularity)에 따라 페이지 수준 매핑(page-level mapping)과 블록 수준 매핑(block-level mapping)으로 나누어진다.

페이지 수준 매핑 기법에서는 매핑의 단위가 플래시 메모리의 읽기, 쓰기 단위인 페이지가 된다. 호스트 시스템으로부터 쓰기 요청을 받을 경우, FTL은 해당 데이터를 미리 지워둔 영역에 기록한 후, 논리 페이지 주소와 물리 페이지 주소의 매핑 테이블 정보를 갱신한다. 읽기 요청을 받을 경우, FTL은 유지하고 있는 매핑 정보를 참조하여 플래시 메모리의 해당 영역을 읽어 호스트 시스템으로 해당 데이터를 전송하게 된다. 제자리 갱신을 허용하지 않는 플래시 메모리의 제약 조건을 피하기 위해 FTL은 플래시 메모리를 파일 시스템 연구에서의 로그 구조 파일 시스템(log-structured file system)과 유사한 방법으로 운영하며, 호스트 쓰기 요청된 데이터는 항상 로그의 제일 앞에 붙여 쓰기 방식으로(append-only manner) 기록된다. 페이지 단위 매핑 기법에서는 논리 페이지가 임의의 물리 페이지에 매핑될 수 있으므로 임의 쓰기(random write) 요청 처리 성능이 우수하다.

페이지 수준 매핑 기법은 매핑의 단위가 작기 때문에 유연성 있고 호스트의 임의 쓰기(random write) 성능이 뛰어나다는 장점이 있지만, 유지해야 하는 매

핑 테이블이 매우 크기 때문에 이를 모두 메모리에 유지하는 것은 부담이 될 수 있다. 이러한 부담을 완화하기 위하여 매핑 테이블 자체도 플래시 메모리에 유지하고 자주 참조되는 소량의 매핑 엔트리만을 메모리에 유지하는 FTL도 제안되었다.

페이지 수준 매핑 FTL에서 논리 주소는 예비 영역에 기록되며, 저장 장치 시스템이 부팅될 때 FTL은 모든 페이지의 예비 영역을 읽어 논리 페이지 주소와 물리 페이지 주소 사이의 매핑 테이블을 메모리에 유지하게 된다. 논리 주소 공간에서 덮어쓰기에 의해 무효화된 페이지를 유효한 최신 페이지와 구분하기 위해서 일반적으로 일련 번호(sequence number) 역시 예비 영역에 기록된다. 이러한 방식의 페이지 매핑 FTL은 부팅 시간이 오래 걸리며 매핑 테이블을 메모리에 항상 유지해야 하는 부담이 크다는 단점이 있다.

DFTL[7]은 페이지 매핑 테이블을 플래시 메모리에 유지하고 자주 접근되는 일부의 매핑 엔트리들만 메모리에 유지함으로써 메모리 요구 사항이 큰 페이지 수준 매핑 기법의 단점을 보완하였다. DFTL 기법에서 플래시 메모리의 전체 공간은 호스트 데이터를 저장하는 데이터 영역(data region)과 맵 영역(map region)으로 나누어지며, 부팅 시에 맵 영역에 있는 페이지들만을 스캔하여 최신의 매핑 테이블을 구축할 수 있다. 일단 매핑 테이블을 구성하는 각 페이지의 위치가 파악된 후에는 호스트 요청에 대해 필요한 엔트리를 포함하는 페이지를 필요에 의해 메모리에 로드하여 처리할 수 있다.

블록 수준 매핑 기법에서는 매핑의 단위가 플래시 메모리의 소거 단위인 블록이 된다. 호스트 시스템으로부터 요청되는 읽기 및 쓰기 요청의 논리 주소는

논리 블록 주소와 해당 논리 불록 안에서의 페이지 주소(offset)로 분리가 되며, 논리 블록 주소는 FTL이 내부적으로 유지하는 블록 매핑 테이블에 의해 물리 블록 주소로 변환된다. 페이지 수준 매핑 기법에 비해 매핑의 단위가 훨씬 큰 블록 단위로 이루어지므로, 매핑 테이블의 크기는 페이지 수준 매핑 기법에 비해 훨씬 작아지고 이를 메모리에 유지하는 부담도 훨씬 적다. 하지만 호스트로부터의 쓰기 요청 데이터를 임의의 페이지에 기록할 수 있는 페이지 수준 매핑 기법과 달리 블록 수준 매핑에서는 페이지의 오프셋이 항상 고정되므로, 단 하나의 페이지를 쓰는 데에도 해당 블록의 나머지 페이지들을 기존의 블록에서 읽어 복사해야 하는 부담이 크다. 결국 대량의 페이지를 기록하는 순차 쓰기(sequential write)에는 유리하지만, 임의 쓰기(random write) 성능이 크게 저하된다. 블록 수준 매핑 FTL에서도 페이지 수준 매핑 기법에서와 마찬가지로 논리 블록 주소와 일련번호를 예비 영역에 기록하고 저장 장치 시스템 부팅 시 예비 영역을 읽어 최신의 매핑 테이블을 메모리에 구축할 수 있다.

블록 안에서의 페이지 오프셋이 고정되어야 하는 제약으로 임의 쓰기의 성능 저하가 크기 때문에 순수한 형태의 블록 수준 매핑 기법은 거의 쓰이지 않으며 다양한 비휘발성 쓰기 버퍼를 도입하는 기법이 제안되어져 왔다. 이러한 쓰기 버퍼를 사용하는 블록 매핑 FTL에서 호스트로부터 쓰기 요청된 데이터는 우선 미리 예비해둔 블록에 기록되며 나머지 데이터를 복사하는 작업은 수행되지 않는다. 쓰기 버퍼에 기록되는 페이지들은 블록 매핑 또는 페이지 매핑으로 처리될 수 있다.

쓰기 버퍼를 이용한 연구 초기에 제안된 대체 블록 기법[18]은 기본 매핑 테이블 뿐 아니라 쓰기 버퍼 역시 블록 매핑을 사용하기 때문에 논리 주소를 논리

블록과 블록 안에서의 오프셋으로 분리했을 경우 오프셋은 쓰기 버퍼에서도 동일한 위치를 가리키게 된다. 이러한 이유로 논리적인 덮어쓰기 요청이 다시 올 경우, 이미 존재하는 대체 블록을 사용하지 못하고 새로운 대체 블록을 할당해 기록하게 된다. 이렇게 하나의 논리 블록에 다수의 물리적인 대체 블록이 리스트 형태로 연결되고, 추후에 블록 병합 과정을 거쳐 주 블록 매핑 테이블에 등록된다. 대체 블록 기법은 순수 블록 매핑보다는 복사의 부담이 적지만 고정된 위치 쓰기 방식으로 인해 여전히 소량의 임의 쓰기 시 성능 저하가 발생한다.

Kim 등에 의해 제안된 로그 블록 기법[4]에서는 주 매핑 테이블은 블록 수준 매핑을 사용하고, 쓰기 버퍼는 페이지 수준 매핑으로 관리한다. 대체 블록 방식에서 논리 주소 공간에 대한 덮어쓰기 요청이 올 경우 새로운 대체 블록이 계속 할당되어져야 하는 것과 달리 로그 블록 기법에서는 쓰기 버퍼를 페이지 매핑으로 관리하기 때문에 하나의 쓰기 버퍼에 계속 추가 저장하는 방법으로 처리 가능하기 때문에 대체 블록 방식에 비해 쓰기 버퍼를 훨씬 효율적으로 사용할 수 있다. 소거 블록이 필요할 경우 블록 병합(block merge) 작업을 수행하게 된다.

로그 블록 기법은 대체 블록 대비 쓰기 블록 버퍼 사용 효율은 훨씬 높아졌지만, 데이터 블록과 쓰기 버퍼가 1:1 로 매핑되기 때문에 호스트 요청이 여러 개의 논리 블록에 걸쳐 요청될 경우 쓰기 버퍼에 가용 페이지가 남아있음에도 불구하고 블록 병합 작업이 강제로 시작되는 상황이 발생할 수 있다. 이를 극복하기 위해 Lee 등에 의해 제안된 FAST(Fully Associative Sector Translation) 기법[5]은 데이터 블록과 쓰기 버퍼 블록간의 임의 매핑을 지원함으로써 보다 높은 쓰기 버퍼 활용율을 얻을 수 있었다.

2.3 쓰레기 수집 작업

새로운 데이터를 덮어 쓰는 것이 불가능하며, 쓰기 가능한 영역을 만드는 소거 작업은 페이지보다 훨씬 큰 블록 단위로 이루어지는 플래시 메모리의 제약 조건으로 인해 쓰레기 수집 작업(garbage collection)이 필요하며 이는 FTL의 주요 역할 중의 하나이다.

쓰레기 수집 작업은 전통적으로 컴퓨터 시스템의 휘발성 및 비휘발성 메모리 자원에 대한 관리를 위해 다양하게 연구 되어왔으며, 연구 분야에 따라 약간씩 다른 의미를 가지고 있다. 휘발성 메모리에서의 쓰레기 수집 작업은, 기존의 프로그래머가 직접 메모리의 할당 및 해제를 명시적으로 표현하던 방식에서 벗어나, 참조 계수(reference counting) 방식을 통해 메모리를 동적으로 할당하고 사용하지 않게 된 영역을 자동으로 해제하는 연구가 이루어져 왔다. 이러한 연구는 주로 프로그래밍 언어 분야에서 많이 수행되어져 왔으며 자바(Java) 등의 객체 지향 언어에서 많이 적용되고 있다. 비휘발성 메모리에서의 쓰레기 수집 작업의 예로는 로그 구조 파일 시스템에서의 세그먼트 클리닝(segment cleaning) 작업을 들 수 있다. 이는 소량의 임의 쓰기 성능에 취약하며 대량의 순차 쓰기 성능이 우수한 하드 디스크 기반 저장 장치에서 쓰기 가능한 대량의 공간(chunk)을 미리 생성하기 위한 쓰레기 수집 작업의 일종이라고 볼 수 있다.

주로 성능 상의 이득을 위해 수행되는 하드 디스크 기반 저장 장치에서의 쓰레기 수집 작업과 달리, 덮어쓰기가 불가능한 플래시 메모리에서의 쓰레기 수집 작업은 소거 작업을 통한 쓰기 가능 영역을 미리 생성하여야 된다는 면에서 필수적인 작업이라고 할 수 있다.

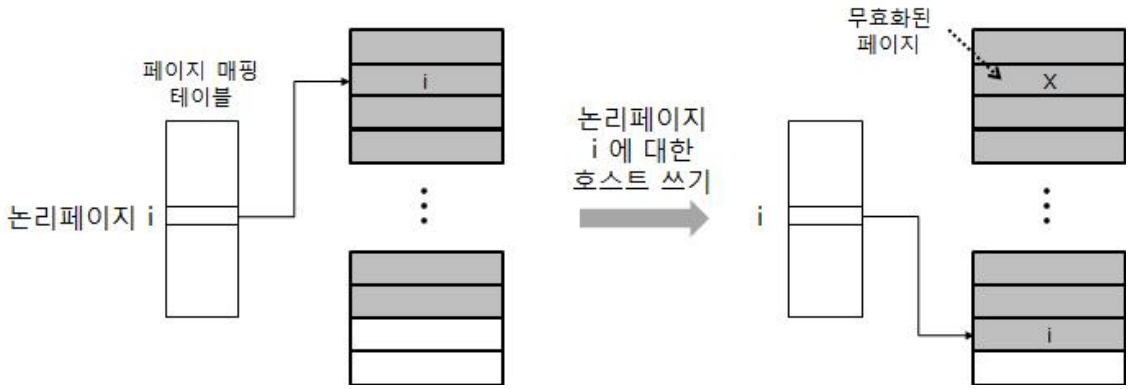

그림 5 호스트 쓰기 처리에 따른 페이지 무효화 과정

플래시 메모리 기반 저장 장치에서는 주소 공간의 일부분에 대해 쓰기 요청이 올 경우, 해당 데이터는 미리 소거되어져 있는 영역에 기록되고 논리 주소와 물리 주소의 매핑이 갱신된다. 이 때, 기존 데이터가 기록되어 있던 영역은 무효화(invalidation)되었다고 한다. 그림 5는 일반적인 페이지 매핑 FTL에서 호스트 쓰기 요청이 처리되는 과정을 나타내고 있다. 논리 페이지 i에 대한 호스트 쓰기 요청은 미리 지워져 있는 물리 영역에 기록되고 매핑 테이블의 엔트리가 새로 쓰여진 물리 페이지를 가리키도록 변경된다. 이 때 기존 데이터가 위치하고 있던 페이지는 무효화 되는 것을 알 수 있다(X로 표시된 부분). 이렇게 무효화된 페이지에는 새로운 데이터를 직접 쓸 수 없으며, 해당 페이지가 속한 블록이 소거 작업이 된 후에야 새로운 쓰기 작업을 처리할 수 있다.

플래시 메모리 기반 저장 장치가 사용됨에 따라 사용 가능한 미리 소거된 페이지의 개수가 줄어들게 되고 이 수치가 정해진 임계값보다 작아지게 되면 쓰레기 수집(garbage collection) 작업이 시작된다. 먼저 소거 작업을 통해 쓰기 가능 공간을 생성하기 위한 희생 블록(victim block)을 선정하게 되며, 이를 위해

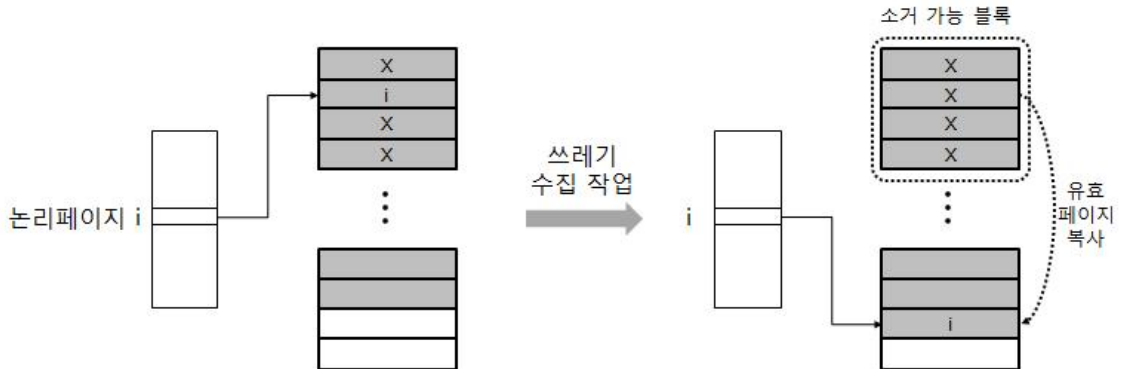

그림 6 페이지 매핑 FTL에서의 쓰레기 수집 작업

일반적으로 FTL은 모든 물리 페이지에 대한 유효성 정보를 비트맵 형태로 유지한다. 희생 블록으로 선정되는 블록은 일반적으로 모든 페이지가 무효화된 블록 또는 유효 페이지의 개수가 가장 적은 블록이 된다. 유효 페이지가 아직 남아있는 블록의 경우 해당 유효 페이지를 로그의 앞부분으로 복사한다. 이렇게 해서 희생 블록의 모든 페이지가 무효화 되면 해당 블록을 소거하여 쓰기 가능한 블록의 자유 공간 리스트에 추가할 수 있게 된다.

그림 6은 일반적인 페이지 매핑 FTL에서 희생 블록으로 선정된 물리 블록 안에 유효한 페이지가 있을 경우 이를 복사하는 작업(valid page copy)을 보여주고 있다. 쓰기 데이터를 호스트에서 받지 않고 기존의 영역에서 읽는다는 점을 제외하면 호스트 쓰기 요청과 동일하다고 할 수 있다. 이 경우 논리 주소는 예비 영역에 미리 기록된 정보를 이용하여 알 수 있다. 그림에서는 유효 페이지가 하나인 경우만 표현하였지만, 일반적으로 복수 개의 페이지를 복사하는 작업이 필요하며, 이러한 유효 페이지 복사 작업에 의해 물리 블록 안의 모든 페이지가 강제적으로 무효화된 이후에 해당 블록은 소거 작업을 거쳐 FTL이 관리하는

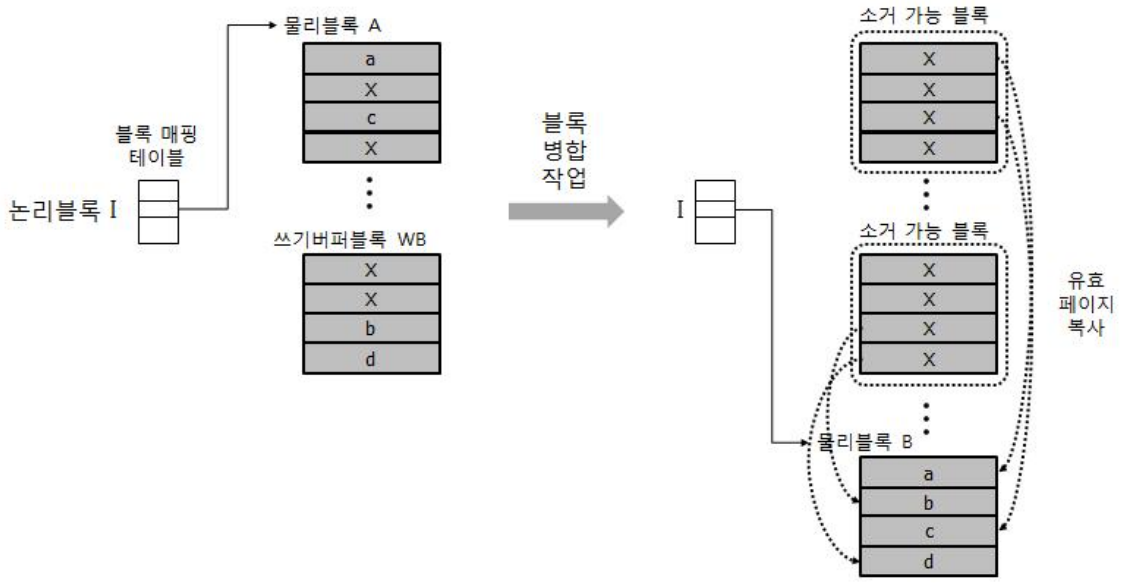

그림 7 블록 매핑 FTL에서의 쓰레기 수집 작업

자유 블록 리스트에 삽입된다.

블록 매핑 FTL에서도 쓰레기 수집 작업은 필요하며, 이는 기존에 매핑되어 있던 블록과 쓰기 버퍼 블록에서 각각 유효한 페이지들을 통합하여 미리 지워진 블록에 기록하기 때문에 일반적으로 블록 병합(block merge) 작업이라고 불린다. 그림 7의 왼쪽은 쓰기 버퍼를 이용하는 블록 매핑 FTL의 예로, 논리 블록 I는 물리 블록 A를 가리키고 있으며, 쓰기 버퍼 블록 WB에는 해당 논리 블록의 페이지 데이터 중 페이지 b와 페이지 d가 기록되어져 있는 것을 보여주고 있다. 쓰기 버퍼는 페이지 매핑으로 관리하기 때문에 논리 페이지는 쓰기 버퍼내의 어느 위치에도 기록될 수 있다. 블록 병합 작업은 물리 블록 A와 쓰기 버퍼 블록 B의 유효한 페이지를 미리 소거되어진 물리 블록 B에 복사한 후 매핑을 변경하고 두 블록(A, WB)을 소거하는 방식으로 이루어지게 된다.

쓰레기 수거 작업은 호스트 요청과 상관없이 FTL 내부의 필요에 의해 발생하는 부가적인 작업이므로 효율적으로 처리하지 못할 경우 호스트 요청의 지연 시간이 크게 증가하게 된다. 쓰레기 수거 작업의 대상이 되는 희생(victim) 블록을 선정할 때 가장 직관적인 방법은 이미 모든 페이지가 무효화된, 또는 유효 페이지를 가장 적게 지니고 있는 블록을 선정하는 것이다. 하지만 이러한 단순한 방법은 지역성이 있는 쓰기 요청에 대해서는 효율성이 저하되는 현상이 발생할 수도 있다.

쓰레기 수집 작업의 효율성을 높이기 위해서도 다양한 연구가 수행되어져 왔다. Wu 등에 의해 제안된 eNVy 기법에서는 쓰레기 수집 작업 시 FIFO와 지역성 모으기를 결합한 방법을 사용함으로써, 지역성이 있는 호스트 요청들에 대해서도 효율적인 쓰레기 수집 작업을 수행하도록 하였다[29]. Kawaguchi 등에 의해 제한된 기법은 로그 구조 파일 시스템의 세그먼트 클리닝(segment cleaning)과 유사하지만 플래시 메모리에 더욱 적합하게 변형된 비용-이득 분석을 통한 쓰레기 수거 정책을 제안하였다[23].

2.4 전원 오류 복구 관련 문제점 및 기술 추세

하드 디스크를 기반으로 하는 기존의 저장 장치 시스템과 비교해 플래시 메모리 기반 저장 장치 시스템에서의 전원 오류에 대한 복구는 일반적으로 더욱 복잡한 작업을 필요로 한다. 이러한 복잡성의 원인은 크게 다음과 같다.

- 첫째, 전원 오류 시에 플래시 메모리에 쓰기 작업이 진행되고 있었을 경우 해당 페이지는 초기 상태 및 원하는 상태 둘 다 아닌 중간 단계의 가비지(garbage) 상태로 남아 있을 수 있다. 일반적으로 하드 디스크의 경우는 쓰기

작업 중 전원 오류가 발생하더라도 해당 섹터는 원자적으로(atomic) 동작을 마치게 된다. 이처럼 전원 오류 시 진행 중인 쓰기 동작에 의해 잔여 효과(residual effect)가 남아있는 경우에도 덮어 쓰기가 가능한 하드 디스크와 달리 플래시 메모리에서는 더 이상의 쓰기 작업을 수행할 수 없다. 결국 FTL 수준에서 이러한 페이지 들을 검출하여 페이지가 속한 블록에 지우기 작업을 수행하기 전까지는 해당 페이지에 쓰기 작업을 요청하지 않도록 시스템 기동 시에 전원 오류 복구 작업이 필요하다.

- 둘째, 전원 오류 복구 작업 진행 중에 다시 전원 오류가 발생할 수 있다. 만약 전원 오류 복구 중에 검출된 정보에 의해 시스템 상태를 갱신하는 쓰기나 지우기 작업을 수행하게 될 경우 이러한 재귀적인 전원 오류 (recursive power failure)에 의해 검사해야 하는 상태(state)의 경우의 수는 급격하게 증가할 수 있다. 기존의 체계적이지 않은 전원 오류 복구 기법에서는 이러한 재귀적인 전원 오류에 의한 영향이 제대로 고려되지 않고 있다.

- 셋째, 플래시 메모리는 하드 디스크와 달리 동작 중 불량 블록이 발생하는 것을 허용한다. 이를 위해 일반적으로 FTL은 플래시 메모리의 일부 영역을 비축(reserve)하고 있다가 불량 블록 발생 시 교체해주는 방법을 사용한다. 이렇게 갱신된 불량 블록 정보를 기록하기 위해 다시 쓰기 동작이 필요하고 이러한 작업 중에도 다시 재귀적으로 전원 오류가 발생할 수 있다.

기존의 FTL 구현들은 일반적으로 각각의 FTL 구현마다 해당 FTL에 의존적이며 체계적이지 않은 방법으로 불량 블록 처리 및 전원 오류 복구를 구현하고 있다. 이러한 접근 방법으로는 최근의 플래시 메모리 기반 저장 장치 시스템

의 기술 변화 추세를 살펴 볼 때 충분한 신뢰성을 확보하는데 어려움이 있다.

제 3 장　HIL 프레임워크 및 쓰레기 수집 작업

본 연구의 기반이 되는 HIL(Hierarchically Interacting set of Logs)은 체계적이고 완전무결한 전원 오류 복구(crash recovery)를 지원하는 FTL 설계 및 구현의 프레임워크이다. HIL 프레임워크는 특정 FTL의 구현에 의존적이지 않으며, 다양한 FTL로의 적용을 위해 보편적인 구성 요소 및 동작만을 가정하여 설계되었다. 즉, HIL 프레임워크에서 제시하는 틀을 통하여 설계된 FTL은 어떠한 형태의 비동기적인 전원 오류 상황에서도 최신의 정확한 저장 장치 상태로 복구될 수 있음을 보장한다.

HIL 프레임워크의 읽기 및 쓰기 동작 시의 전원 오류 복구의 정확성 증명[13]이 이루어진 바 있으며, 본 연구는 이를 확장하여 쓰레기 수집 동작 시의 전원 오류 복구의 정확성을 보이는 것을 목표로 한다. 본 장에서는 쓰레기 수집 작업을 위한 구성 요소 및 동작이 포함된 HIL 프레임워크에 대해서 기술한다.

3.1 FTL 추상화

HIL 프레임워크가 목표로 하는 것은 다양한 FTL에 적용 가능한 FTL 구현에 독립적인 전원 오류 복구 방법을 제공하는 것이다. 이를 위해서는 우선 다양한 FTL에서 공통적인 구성 요소 및 동작 방식을 파악하여 추상화하는 작업이 필요하다.

플래시 메모리를 저장 매체로 사용하는 데 있어 가장 큰 제약이 되는 것은 제자리 갱신이 불가능하다는 것이다. 이러한 제약 조건에 의해 일반적으로 FTL은 쓰기 요청된 데이터를 이미 지워져 있는 자유 영역에 기록하고 해당하는 매핑

정보를 변경하는 방법을 사용한다. NAND 플래시 메모리에는 제자리 갱신이 불가능하다는 것 외에도 해당 페이지가 속한 블록 안에서는 페이지 번호가 증가하는 순서로 쓰기 작업을 해야 한다는 제약 조건(ascending order page program rule)이 있다. 이러한 제약 조건들에 의해 FTL 들은 일반적으로 붙여 쓰기(append-only write)를 처리하는 로그 구조(log structured) 형태로 구현된다. 일반적으로 FTL은 호스트로부터의 읽기 및 쓰기 요청을 로그에 대한 읽기 및 붙여 쓰기 동작으로 처리하는 것으로 볼 수 있다.

이 경우에 FTL은 호스트 데이터의 논리 주소와 물리 주소 간의 매핑 정보를 메모리 상에 유지하여 호스트로부터의 읽기 동작을 처리하게 된다. 호스트 쓰기 동작 처리 시에는 새롭게 기록되는 데이터에 의해 이미 예전에 기록된 데이터가 무효화(invalidate)된다. 쓰레기 수집 작업 또는 블록 병합 작업을 위해서는 논리 주소 공간에 대한 덮어쓰기로 인한 유효 페이지와 무효 페이지에 대한 정보 또한 메모리에 유지하여야 한다. 이러한 매핑 정보 및 페이지의 유효성(liveness) 정보를 FTL 메타데이터라고 할 수 있으며, FTL 메타데이터 역시 플래시 메모리에 비휘발성으로 기록될 수 있다. 페이지 수준 FTL의 경우는 호스트 데이터를 플래시 메모리 페이지에 기록할 때 해당 페이지의 예비 영역에 논리 주소를 기록하고 부팅 시에 전체 영역을 스캔하여 메모리 상에 이러한 매핑 정보를 재구성하는 방법이 주로 사용되었지만 플래시 메모리 기반 저장 장치의 용량이 급속도로 커지고 있는 최근 기술 추세로 볼 때 부팅 시에 스캔 작업을 하는 것은 비현실적이이라고 할 수 있다. 일반적으로 보면 호스트로부터의 데이터를 기록하는 데이터 로그 외에도 매핑을 기록하는 로그 및, 페이지 유효성 정보를 기록하는 로그가 필요하게 된다.

이렇게 플래시 저장 장치 시스템을 구성하는 다양한 로그들 간에는 계층 구조가 존재한다. 호스트 데이터가 플래시 메모리에 기록되면 변경된 논리 주소와 물리 주소의 매핑을 반영하기 위해 호스트 데이터의 매핑을 관리하는 매핑 테이블에 매핑 엔트리 변경이 가해진다. 이러한 데이터 로그의 매핑 테이블 역시 그 크기가 클 경우 다시 플래시 메모리에 로그의 형태로 저장될 수 있다. 이러한 매핑 로그에 대한 쓰기 요청에 의해 변경되는 위치 정보는 상위의 매핑 로그(매핑의 매핑)에 의해 관리된다. 결국 FTL을 구성하는 각각의 로그들 간에는 매핑 관계에 의해 계층 구조가 생기며, 이러한 로그들 간의 상호 작용을 통해 호스트 요청을 처리하게 된다.

플래시 기반 저장 장치 시스템은 초기의 단일 칩, 단일 프로세서 기반의 단순한 환경에서 최근에는 다수의 플래시 메모리 칩과 다수의 플래시 메모리 컨트롤러의 병렬성(parallelism)을 이용하는 환경으로 변하고 있으며, 그에 따라 큰 성능 향상이 가능해졌다. 호스트 시스템으로부터의 쓰기 및 읽기 요청도 NCQ 등을 지원하는 최신의 호스트 프로토콜을 통해 병렬성이 커지고 있다. 이를 효율적으로 처리하기 위해서는 각각의 로그를 다중 쓰레드(multi-thread)로 구현하는 것이 효과적이다. 즉, 플래시 저장 장치 시스템을 구성하는 플래시 메모리 칩, 플래시 메모리 제어기, 프로세서 및 호스트 시스템의 워크로드의 병렬성을 가정할 수 있다.

앞에서 설명한 것처럼, HIL 프레임워크에 의해 설계된 FTL은 호스트 데이터 및 FTL 메타데이터를 계층 구조를 가지는 로그의 집합으로 구성하고, 각각의 로그 들은 쓰레드로 이루어져 메시지를 통해 서로 상호 작용하는 모델로 추상화 한다.

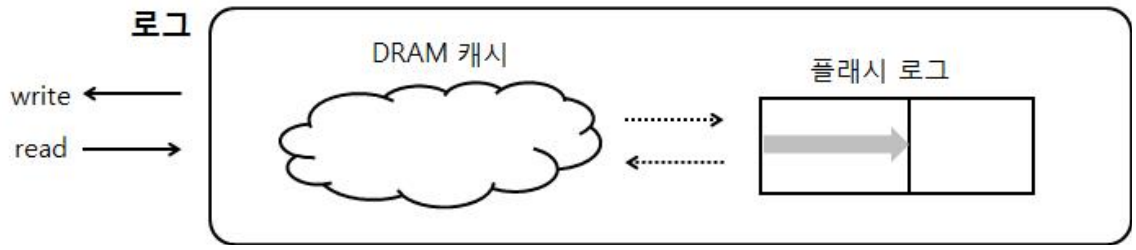

그림 8 HIL 프레임워크에서의 로그 동작 및 내부 구조

3.2 HIL 프레임워크 설계

본 절에서는 앞에서 설명한 추상화된 FTL을 기반으로 하여, HIL 프레임워크의 핵심 구성 요소 및 동작에 대해 설명한다.

3.2.1 HIL 프레임워크 구성 요소: 로그

HIL 프레임워크에서 FTL을 구성하는 핵심 구성 요소인 로그는 그림 8 에서 보이는 것처럼 내부적으로 데이터의 영속성을 제공하기 위한 플래시 로그와 읽기 및 쓰기 성능 향상을 위한 캐시로 구성된다.

플래시 로그는 데이터 로그, 매핑 로그 등 플래시 저장 장치를 구성하는 로그들에 각각 개별적으로(disjoint) 할당된 블록들의 집합이다. 플래시 로그는 항상 페이지를 붙여 쓰기(append-only write)로 기록하며 뒤에서 설명하는 전원 복구 기법을 적용하기 위해 쓰기 순서가 항상 정해져 있어야 한다. 이를 위해 로그의 최전방의 복수 개의 블록에 대한 쓰기 순서를 나타내는 블록 프로그램 리스트라는 자료구조가 사용되며, 이는 주기적으로 체크포인트 로그에 기록된다.

쓰기 요청을 받을 때마다 직접 플래시에 기록할 경우 성능 저하가 매우 크게

된다. 특히 데이터 로그를 제외한 FTL 메타데이터를 기록하는 로그로의 쓰기 요청들의 크기는 플래시 메모리의 페이지 크기에 비하여 매우 작기 때문에 다수의 쓰기 요청을 버퍼링 하여 한꺼번에 쓰는 것이 효율적이다. 이를 위해 로그의 내부에 DRAM등의 휘발성 메모리를 이용하여 캐시를 구성하는 것을 가정하며, 이는 읽기 성능을 향상시키는 용도로도 사용된다. 캐시로 사용할 수 있는 메모리의 크기에는 제약이 있기 때문에 일반적인 LRU 교체 알고리즘 등을 사용하여 필요한 데이터를 플래시 로그에서 가져오거나 오래된 데이터를 플래시 메모리로 플러시 하게 된다.

HIL 프레임워크에서는 기본적으로 다음 네 종류의 로그 타입이 존재한다. 첫째, 호스트 데이터를 저장하기 위한 데이터 로그(D), 둘째, 하위 로그에 대한 매핑 정보를 저장하기 위한 매핑 로그(M), 셋째, 쓰레기 수거 작업을 위한 블록 및 페이지의 상태 정보를 유지하는 물리 블록 정보 로그(L), 넷째, 전원 오류 복구 작업을 위한 다양한 정보를 유지하는 체크포인트 로그(C)가 있다. HIL 프레임워크는 위와 같은 기본적인 타입의 로그들을 필요에 의해 조합하여 다양한 FTL을 구성할 수 있다.

저장 장치 외부의 호스트로부터 데이터 로그로의 쓰기 요청이 오게 되면 해당 페이지는 먼저 DRAM 으로 구성된 캐시에 보관된다. 추후에 일정시간이 지나거나 캐시를 플러시 시킬 필요가 생길 경우 플래시 로그에 영속적으로 기록한 후 쓰기 포인터를 전진시키게 된다.

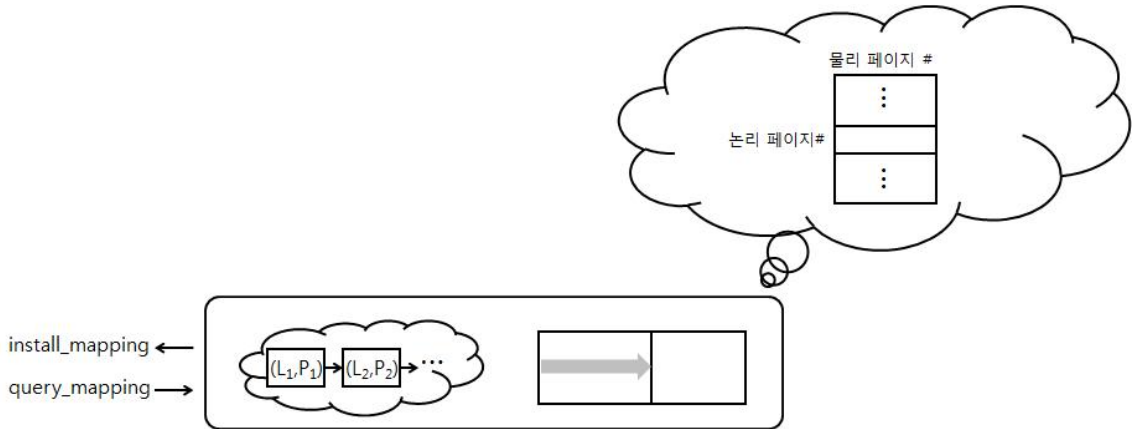

그림 9 매핑 로그의 동작 및 내부 구조

그림 9는 매핑 로그들에서의 동작을 보여준다. 매핑 로그는 하위 로그를 구성하는 페이지들의 논리 주소와 물리 주소의 매핑 테이블을 유지하는 로그로 하위 로그로부터의 매핑 업데이트(install) 요청 또는 매핑 질의(query) 요청을 처리하게 된다. 휘발성으로 유지하는 캐시에는 매핑 정보를 보관하고 자신의 캐시 운영 정책에 따라 추후에 플래시 로그로 기록하게 된다.

그림 10은 물리 블록 정보 로그(liveness log)의 구성을 보여준다. 플래시 저장 장치가 사용됨에 따라 새로운 페이지가 기록되면 기존에 매핑 되어져 있던 페이지는 무효화가 되며 이렇게 무효화된 페이지만으로 이루어진 물리 블록은 소거되어 자유 블록 리스트에 들어가게 된다. 이를 위해서는 각각의 물리 페이지에 대한 유효성 정보가 필요하며 비트맵 형태로 물리 블록 정보 로그에 의해 관리 및 기록된다. 또한, 희생 블록을 선택할 때에는 페이지의 무효화 정도 외에도, 마모도 평준화 작업을 위해 물리 블록의 현재까지의 소거 횟수를 고려할 수 있으며, 이러한 소거 횟수 정보 역시 물리 블록 정보 로그에 유지된다. 이러한

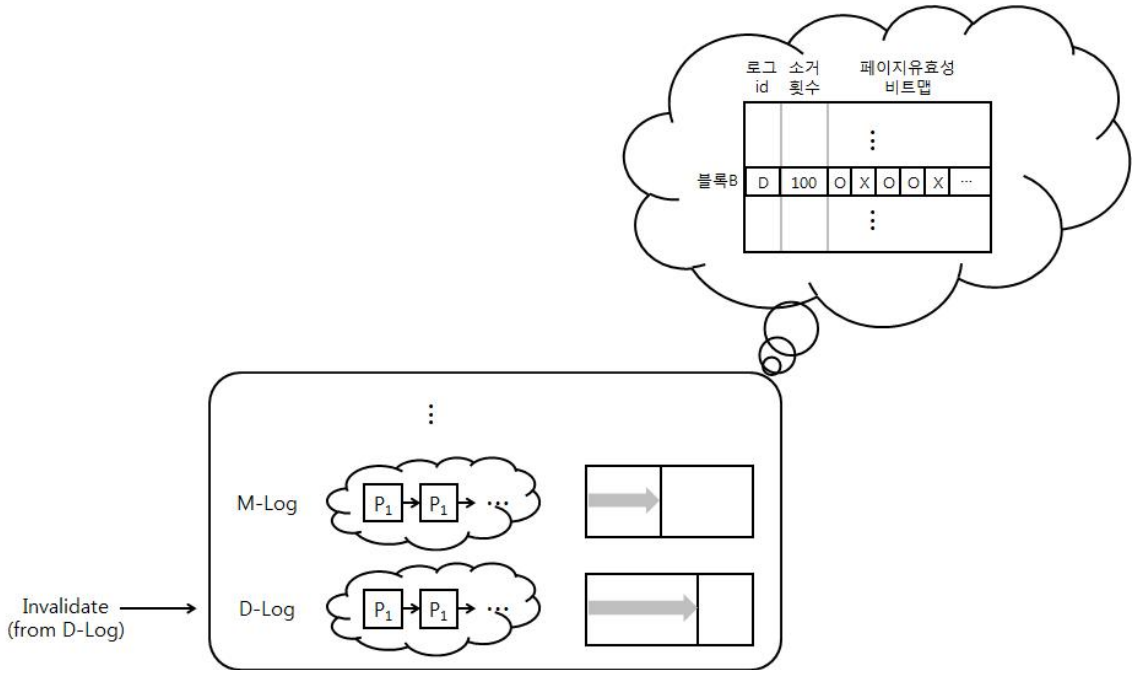

그림 10 물리 블록 정보 로그의 동작 및 내부 구조

정보는 전역적으로 유지되므로 물리 블록 정보 로그는 모든 로그로부터 요청을 받는 형태가 되며, 이에 따라 캐시 구성 역시 각각의 로그에 따라 별도의 캐시를 사용한다. 하나의 물리 블록은 사용 중에는 하나의 로그에만 속해야 하며, 쓰레기 수집 작업은 해당 로그에서 처리하게 된다. 이를 위해 물리 블록의 소속 정보를 기록하는 로그 아이디 정보 역시 기록하게 된다.

3.2.2 로그 간의 계층 구조와 상호 작용

앞 절에서 설명한 바와 같이 호스트에 의해 쓰기 요청되는 데이터를 저장하기 위한 데이터 로그 뿐 아니라 이렇게 기록된 데이터들에 의해 변경되는 매핑 및 유효성 정보 역시 플래시 메모리에 영속적으로 기록되어져야 한다.

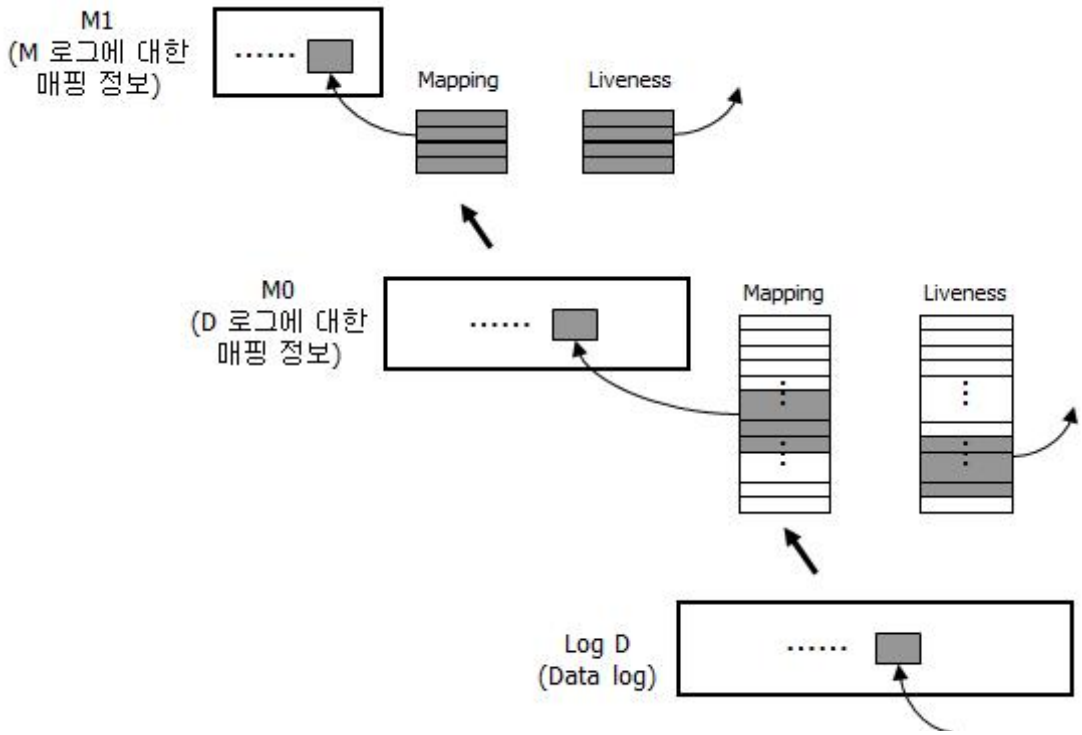

그림 11 로그 들 간의 계층 구조

그림 11은 일반적인 페이지 수준 FTL을 구성하는 로그들 간의 계층 구조를 보여주고 있다. 페이지 수준 FTL은 매핑 정보의 크기 역시 크기 때문에 하나의 플래시 메모리 페이지로 모든 정보를 기록할 수 없다. 결국 매핑 정보를 기록하는 로그(M0)는 해당 로그의 논리 공간에 대한 매핑(매핑의 매핑) 정보를 기록하는 또 다른 상위의 로그(M1)가 필요할 수 있다. 최종적으로 매핑의 모든 정보를 하나의 페이지로 기록할 수 있는 수준이 되면 더 이상의 상위 로그를 이용한 매핑은 필요하지 않게 된다. 매핑 로그 뿐 아니라 물리 블록 정보 로그 역시 관리하는 정보의 양이 클 경우, 유사하게 여러 단계의 로그로 구성할 수 있다.

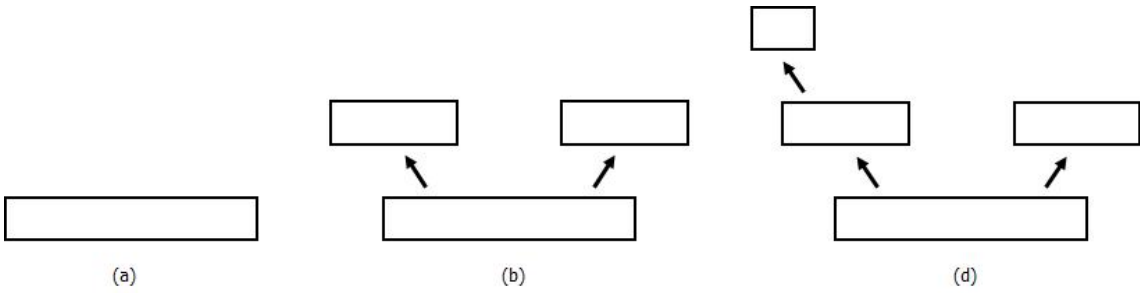

그림 12 로그 조합을 통한 다양한 FTL 구성

3.2.3 로그 조합을 통한 FTL 구성

그림 12는 로그의 조합을 통해 구성할 수 있는 다양한 FTL의 예를 보여준다. 그림 12-(a)는 하나의 데이터 로그로만 구성된 FTL의 구성 예이다. 매핑 정보를 따로 저장하지 않고 예비 영역에 논리 주소를 기록하는 저용량의 페이지 수준 FTL의 경우 이러한 단일 로그로 구성이 가능하다. 이 경우 저장 장치 시스템 기동 시 플래시 메모리의 전체 물리 영역을 스캔하여 매핑 및 페이지 유효성 정보를 메모리 상에 재구성해야 하기 때문에 부팅 시간이 길어지게 된다. 그림 12-(b)의 구성은 이러한 단점을 극복하기 위해 데이터 로그에 대한 매핑 정보를 기록하는 별도의 로그를 사용하는 FTL의 예이다. 이 경우 전체 매핑 정보는 상위의 한 단계의 매핑 로그를 읽는 것만으로 가능해진다. 페이지의 유효성 정보 역시 별도의 로그로 기록할 수 있다. 그림 12-(c)의 구성은 보다 일반적으로 매핑 정보 역시 상위의 매핑 정보를 가질 수 있음을 보여준다. 대용량의 페이지 레벨 FTL의 경우 매핑 테이블의 크기가 매우 커질 경우 이와 같이 여러 단계의 매핑 로그들을 이용한 구성으로 구현하는 것이 가능하다.

3.3 HIL 프레임워크 동작

블록 인터페이스를 제공하는 저장 장치 시스템은 호스트 시스템에게 쓰기 및 읽기가 가능한 섹터 기반의 논리 주소 공간을 제공한다. 쓰기 요청된 데이터 각각에 대해서 비휘발성으로 저장 매체에 기록을 하는 것은 매우 비효율적이기 때문에 일반적으로 저장 장치 내부에 존재하는 DRAM 버퍼에 휘발성으로 저장된 후에 처리가 완료되었음을 호스트 시스템에게 알려주게 된다. 이렇게 DRAM 버퍼에 임시로 저장된 데이터는 추후에 호스트로부터 명시적인 플러시 요청을 받거나, 저장 장치 내부의 캐시 교체 정책에 의해 필요한 경우 플래시 로그에 기록된다. 호스트 읽기 요청 역시 DRAM 캐시에 있을 경우 바로 처리가 가능하며, 그렇지 않을 경우 플래시 로그에서 해당 페이지를 읽어 처리하게 된다. 호스트로부터 명시적으로 요청되는 쓰기 및 읽기 요청 외에도 플래시 기반 저장 장치 시스템은 쓰기 가능 공간을 만들어 내기 위한 쓰레기 수집 작업이 필요하다. 이는 물리 블록 정보 로그의 페이지 유효성 정보를 이용하여 유효 페이지 복사 및 블록 소거 작업을 통해 이루어진다.

본 절에서는 HIL 프레임워크에 의해 설계된 플래시 메모리 기반 저장 장치 시스템에서의 호스트로부터의 쓰기 및 읽기 동작 처리와 쓰레기 수집 작업에 대해서 자세하게 설명한다.

3.3.1 호스트 쓰기 요청 처리

그림 13은 HIL 프레임워크 기반의 FTL에서 호스트로부터의 섹터 단위의 쓰기 요청을 처리하는 과정을 보여준다. 섹터 단위의 호스트 요청은 우선 쓰기 요청을 처리하는 쓰기 쓰레드(writer thread)에게 전달되고, 페이지 단위의 요청

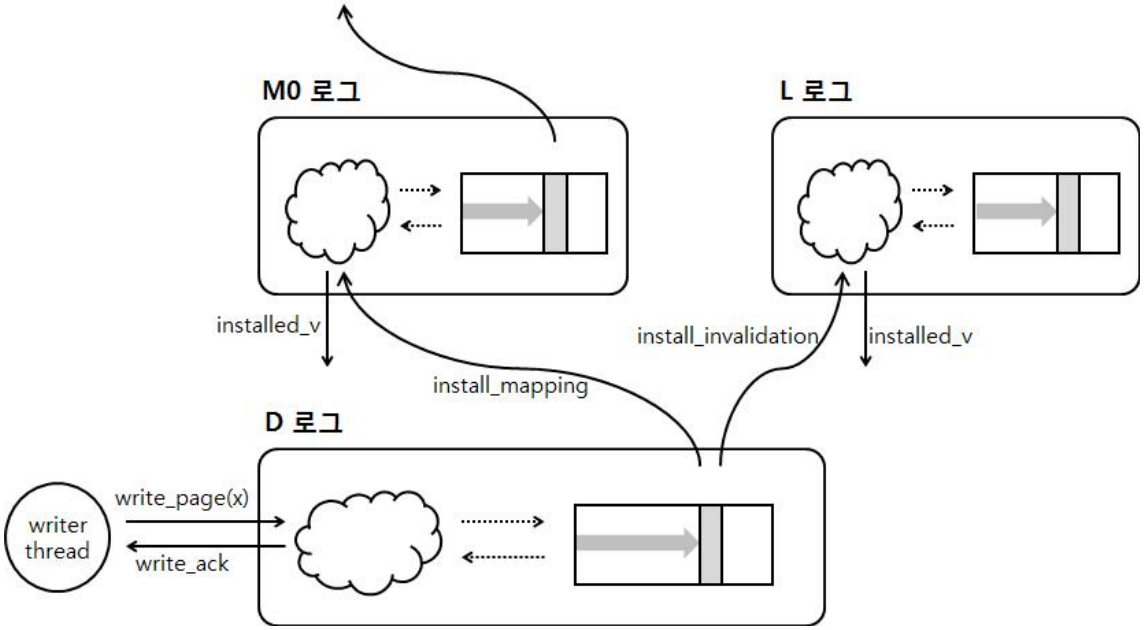

그림 13 HIL 프레임워크에서의 호스트 쓰기 요청 처리 과정

으로 분할되어 데이터 로그(D 로그)에게 다시 전달된다. 데이터 로그에 전달된 쓰기 요청이 내부의 휘발성 버퍼에 저장되면 쓰기 응답을 쓰기 쓰레드로 반환하게 된다. 이렇게 호스트 쓰기 요청에 의해 발생하는 모든 페이지 데이터가 데이터 로그 안의 DRAM 버퍼에 저장된 후에 쓰기 쓰레드는 호스트 시스템으로 쓰기 요청 완료 응답을 보낼 수 있고, 버퍼에 저장된 호스트 데이터는 추후에 호스트 시스템으로부터의 버퍼 비우기 명령 또는 로그 자체의 버퍼 관리 정책에 의해 플래시 로그에 비휘발성으로 저장(flush)된다.

매핑 관계를 살펴보면, 데이터 로그의 플래시 로그에 새로 붙여 쓰기(append)된 데이터에 의해 상위 매핑 로그로의 매핑 갱신 요청이 필요하게 된다. 논리 주소 X의 데이터가 물리 주소 p에 기록되었다는 정보가 install_map-

ping(X, p)의 형태로 상위 매핑 로그인 M0에 전송된다. 이러한 요청은 M0의 비휘발성 버퍼에 저장된 것이 확인되면 처리가 끝나게 되며 비휘발성으로 기록이 완료되었다는 의미의 installed_v 메시지로 응답을 해주게 된다. 개념적으로 살펴보면, 데이터 로그는 페이지 단위의 쓰기 및 읽기를 지원하고, 매핑 로그는 매핑 엔트리 단위의 쓰기 및 읽기를 지원한다는 차이가 있지만 기본적인 동작은 동일하다는 것을 알 수 있다. 즉 write_page 메시지는 install_mapping 요청과 대응되며 write_ack 메시지는 installed_v 응답과 대응된다. HIL 프레임워크에서는 이와 같이 FTL을 구성하는 모든 로그를 유사한 방법으로 추상화하여 설계하였기 때문에 구현의 부담이 적고 다양한 FTL을 표현할 수 있으며 정확성 검증에 필요한 경우의 수를 최소화할 수 있게 된다.

M0 로그의 버퍼에 저장된 매핑 정보는 추후에 버퍼 관리 정책에 의해 플래시 로그에 비휘발성으로 저장되며, 이는 M0 로그의 논리 주소 공간에 대한 쓰기가 처리된 것이기 때문에 다시 상위의 M1 로그로 install_mapping 요청이 전송된다. 이렇게 재귀적으로 반복되는 각 단계의 로그에서의 쓰기(매핑 갱신) 요청은 모든 요청이 한 페이지를 기록하는 것으로 가능해지는 최상위 로그까지 전송되면 처리가 모두 끝나게 된다.

페이지 유효성(liveness)관점에서 살펴보면, 새롭게 기록된 호스트 데이터에 의해 기존에 논리 주소 X에 매핑되어 있던 물리 페이지는 무효화 되며, 이러한 사실을 물리 블록 정보 로그에 전달하여야 한다. 즉, 물리 블록 정보 로그가 관리하는 페이지 비트맵 정보에 대한 갱신 요청이 필요하며, 이는 install_invalidation(q)의 형태로 물리 블록 정보 로그 L에 전달된다. L 로그 역시 내부에 존재하는 DRAM 버퍼에 이 요청이 저장된 후에 바로 installed_v 형태로 응답할 수

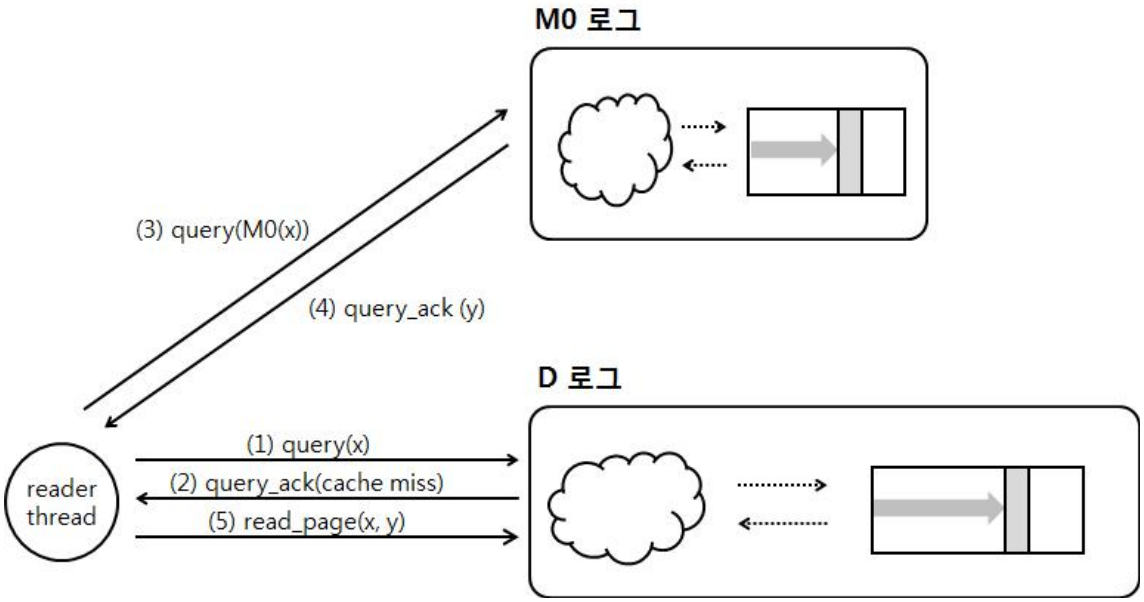

그림 14 HIL 프레임워크에서의 호스트 읽기 요청 처리 과정

있고 추후에 자신의 플래시 로그에 영속적으로 기록하게 된다.

3.3.2 호스트 읽기 요청 처리

그림 14는 HIL 프레임워크 기반의 FTL에서 호스트로부터의 섹터 단위의 읽기 요청을 처리하는 과정을 보여준다. 호스트 쓰기 요청 처리와 유사하게 호스트 읽기 요청은 읽기 쓰레드를 거쳐 페이지 단위의 질의(query) 요청으로 변환되어 데이터 로그로 전달된다. 데이터 로그의 논리 주소 공간 X에 대한 읽기 요청은 데이터가 데이터 로그 내의 캐시에 존재하는 경우 이를 이용해 호스트로 해당 데이터를 전송하고 즉시 처리를 종료할 수 있다. 로그 내의 캐시에 해당 데이터가 존재하지 않는 경우에는 읽기 쓰레드로 cache_miss 응답이 반환된다. 읽기 쓰레드는 해당 데이터가 저장되어 있는 플래시 메모리의 물리 페이지 주소

를 얻기 위해 상위 로그 M0 에 다시 질의 query(M0(X))를 요청하게 된다. 논리 주소 X에 대한 물리 주소의 매핑 엔트리가 M0 로그의 캐시에 존재할 경우는 query_ack(y) 을 통해 해당 물리 주소를 반환하게 되고 읽기 쓰레드는 물리 주소에 해당하는 데이터를 다시 데이터 로그에 read_page 요청을 통해 물리 주소와 함께 전송하여 호스트의 읽기 요청을 처리하게 된다.

3.3.3 쓰레기 수집 동작 절차

플래시 메모리 기반의 저장 장치는 덮어쓰기가 불가능한 플래시 메모리의 특성에 의해 호스트로부터의 쓰기 요청은 이미 소거되어진 페이지에 쓰여지고 해당 논리 주소에 매핑되어져 있던 영역은 무효화가 된다. 사용 시간이 증가함에 따라 쓰기 가능한 공간은 줄어들고 무효화된 영역의 크기가 증가하게 된다. FTL은 내부적으로 희생 블록을 선정하고 유효 페이지들을 로그의 앞으로 복사하여 강제로 희생 블록 전체를 무효화 시킨 후 소거 작업을 거쳐 쓰기 가능한 공간으로 재생산하게 된다.

그림 15는 HIL 프레임워크에서 쓰레기 수집 동작이 이루어지는 과정을 보여준다. 쓰레기 수집 동작은 호스트 시스템으로부터 명시적으로 요청되는 호스트 쓰기 및 읽기 동작과 달리 FTL 내부의 필요에 의해 개시된다. 즉, 어느 시점에 로그에 대한 쓰기 요청을 처리할 수 있는 소거 블록의 개수가 임계치보다 작아지면 쓰레기 수집 작업이 시작된다.

쓰레기 수집 작업은 물리 페이지의 유효성 정보를 관리하는 물리 블록 정보 로그(L 로그)가 담당하게 된다. 희생 블록을 선정하는 방법은 여러 가지가 있을 수 있지만 가장 단순한 정책은 유효 페이지가 가장 적은 물리 블록을 선정하는

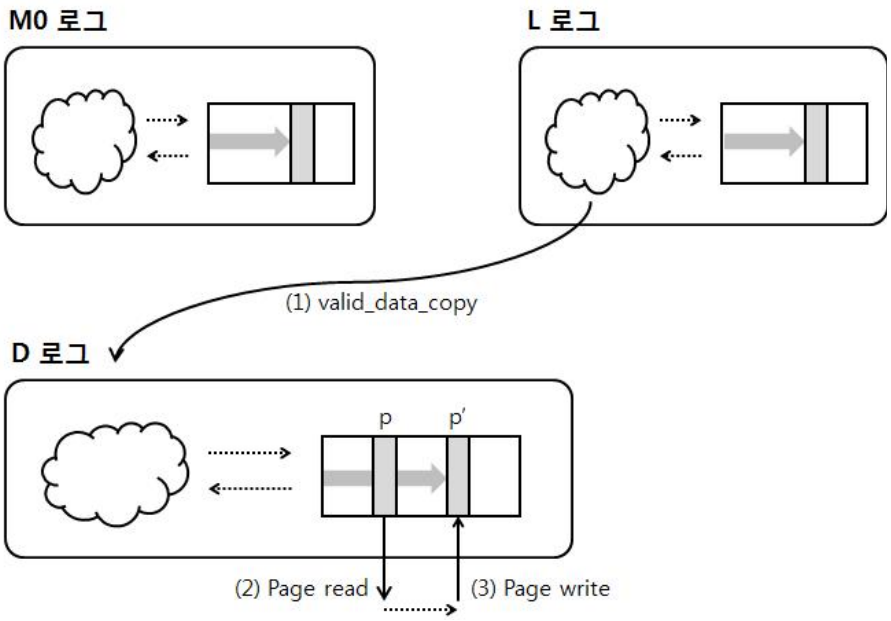

그림 15 HIL 프레임워크에서의 쓰레기 수집 동작 절차

것이다. 만약 모든 페이지가 무효화된 블록이 있다면 해당 블록을 소거하고 쓰기 가능한 자유 블록 리스트에 삽입할 수 있다. 유효 페이지가 하나라도 있는 블록을 선택했다면, 해당 유효 페이지 들을 강제적으로 무효화시키는 것이 필요하다. 물리 블록 정보에는 유효성 정보 뿐 아니라 해당 물리 블록이 속해 있는 로그의 아이디가 같이 기록되며, 이를 이용하여 유효 페이지 복사(valid page copy) 요청을 해당 로그로 전송하게 된다. 그림에서는 데이터 로그로 요청이 전달된 것을 보여주며, 이 경우 데이터 로그는 해당 물리 주소를 이용하여 플래시 페이지를 읽은 후 로그의 최전방에 다시 기록하게 된다. 이 때 페이지를 플래시 로그에 다시 기록하는 것은 호스트의 쓰기 요청 처리와 동일하다고 볼 수 있다. 페이지가 쓰여진 후에는 M0 로그 및 L 로그로 install_mapping 과 install_invalidation 요청이 전송된다. 결국 L 로그에서 받은 무효화 요청에 의해 해당 물

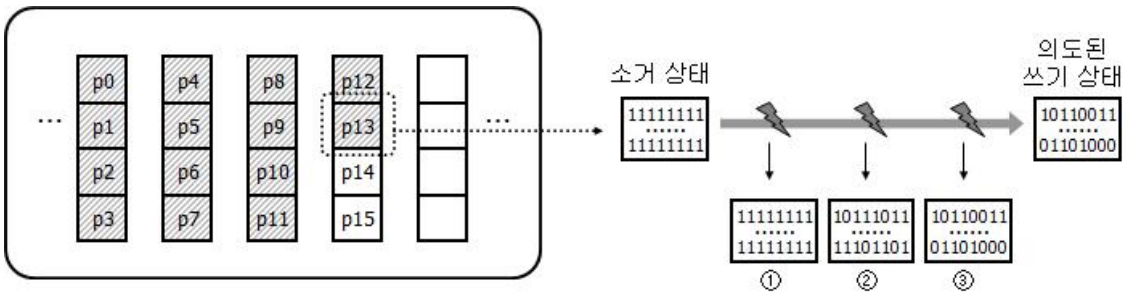

그림 16 프로그램 중 전원 오류 발생에 따른 페이지 상태

리 블록의 모든 페이지가 무효화 되고 소거 작업을 거쳐 자유 블록 리스트에 삽입되는 과정을 거쳐 쓰레기 수집 작업을 마치게 된다.

3.4 HIL 프레임워크에서의 전원 오류 복구

HIL 프레임워크에서의 전원 오류 복구는 구조적 복구와 기능적 복구의 두 단계를 통해 이루어진다. 구조적 복구는 쓰기 동작 중 전원 오류가 발생하는 경우 해당 페이지에 남겨지는 잔여 효과를 제거하여 쓰기 가능한 물리적인 로그 구조를 복구하는 단계이며, 기능적 복구는 예비 영역에 저장된 재현 정보를 통하여 저장 장치의 상태를 전원 오류가 발생한 시점 이전의 최신의 일관된 상태로 복구하게 된다.

3.4.1 구조적 복구 (structural recovery)

그림 16은 플래시 메모리에 대한 페이지 쓰기 동작 중에 전원 오류가 발생할 경우 해당 페이지의 상태를 보여준다. 플래시 메모리는 전원 오류에 대해 원자적(atomic)인 프로그램을 지원하지 않기 때문에 페이지 상태는 크게 i) 모든 비트가 1로 남아있는 상태(①), ii) 일부 의도된 비트만 0으로 변한 상태(②), iii)

모든 의도된 비트가 0으로 변한 상태(③)의 세 가지 경우가 있게 된다. 상태 ③인 경우의 페이지는 의도된 동작이 완료된 것으로 볼 수도 있지만, 메모리 셀의 상태가 안정적이지 않기 때문에 시간이 지난 경우 비트 값이 반전되어 데이터 오류가 발생할 수 있다. 그러므로 ①부터 ③까지의 모든 페이지를 전원 오류에 의한 잔여효과가 발생한 페이지로 간주하여 처리하여야 한다. HIL 프레임워크의 구조적 복구 작업은 이러한 잔여 효과를 제거하는 역할을 수행한다. 이를 위해 각각의 로그마다 무해석 블록(don't-care block)이라 명명된 하나의 블록을 미리 할당하고 저장 장치 시스템 기동 시에 잠재적 오류 가능성이 있는 페이지를 포함하는 블록을 찾아 무해석 블록과 교체하는 방법을 사용한다.

잠재적 오류의 가능성이 있는 페이지는 로그의 블록 프로그램 리스트 순서로 페이지를 스캔하여 찾을 수 있다. 페이지를 순서대로 스캔하며 소거된 페이지 또는 ECC 오류가 발생하는 페이지를 만나게 되면 해당 페이지가 속해 있는 블록에 쓰기 동작 중 전원 오류가 발생했을 가능성이 있다고 할 수 있다. 이러한 블록을 전원 오류 블록(crash frontier block)이라고 명명하고, 전원 오류 블록과 무해석 블록의 교체를 통해 잔여 효과를 제거한다.

그림 17은 블록 교체를 통해 잔여 효과를 제거하는 작업 과정을 보여준다. 여기서 잔여 효과가 발생한 페이지는 블록 B의 세 번째 페이지로 앞의 두 페이지는 유효한 데이터를 지니고 있기 때문에 예비 블록 R로 복사한 후 블록 B와 블록 R을 교체하게 된다.

HIL 프레임워크에서 모든 플래시 메모리로의 쓰기 작업은 이미 존재하는 로그에 붙여 쓰기 형태로만 처리되므로 잔여 효과는 항상 로그의 끝에서 발생한

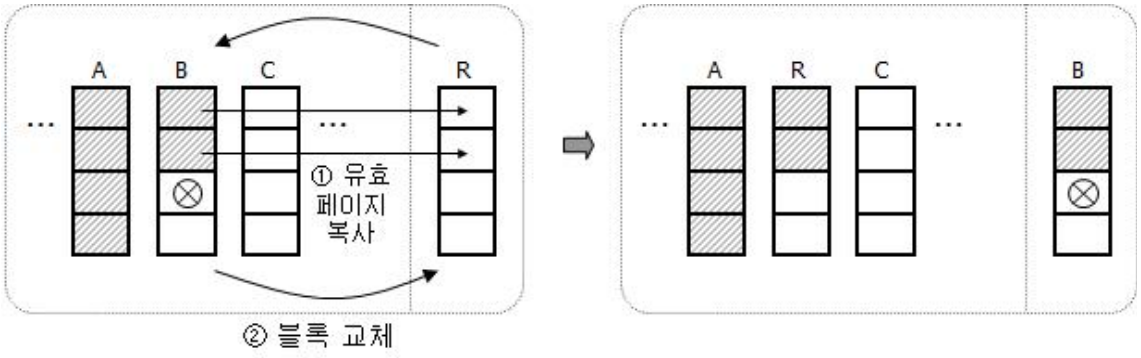

그림 17 무해석 블록을 이용한 잔여 효과 제거

다. 또한 각각의 로그는 병렬적으로 동작할 수 있기 때문에 다수의 로그에 잔여 효과가 남아있을 수 있다. HIL 프레임워크는 각각의 로그에서 잔여 효과를 모두 제거하는 것이 필요하다.

잔여 효과를 제거하는 과정에서 주의해야 할 점은 이러한 잔여 효과를 제거하는 작업 수행 중에 다시 전원 오류가 발생할 수 있다는 사실이다. 이렇게 재귀적인 전원 오류가 발생할 경우 고려해야 할 플래시 메모리의 상태가 무한대로 많아질 수 있기 때문에 HIL 프레임워크에서는 상태를 변경하는 모든 소거 및 쓰기 작업을 무해석 블록만을 이용해서 수행하고 루트 체크포인트 로그에 변경되는 각 로그의 블록 프로그램 리스트를 저장하는 순간 FTL의 전체 상태를 원자적으로 변경하는 방법을 사용한다.

3.4.2 기능적 복구 (functional recovery)

구조적 복구의 단계를 거치고 나면, HIL 프레임워크를 구성하는 각각의 로그 안에 잠재적으로 존재할 수 있는 잔여 효과는 모두 없어지고 로그에 새로운

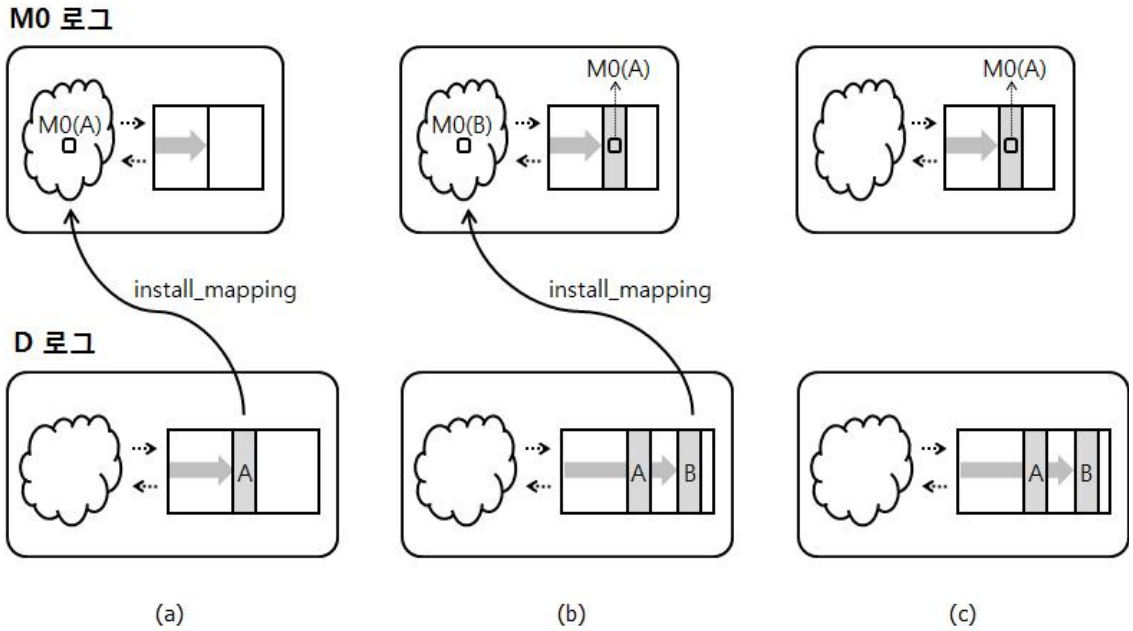

그림 18 전원 오류 발생 시 최신 정보의 소실 예

데이터를 정상적으로 기록할 수 있는 물리적 상태로 복구된다. HIL 프레임워크 전원 오류 복구의 두 번째 단계인 기능적 복구는 플래시 저장 장치의 논리적 상태를 전원오류 발생 직전의 최신의 일관된 상태로 복구한다.

HIL 프레임워크를 구성하는 로그들은 각각 병렬적으로 동작하며, 성능 향상을 위해 내부적으로 DRAM 캐시를 이용하기 때문에 전원 오류가 발생하면 최신 정보가 소실될 수 있다. 그림 18은 데이터 로그와 매핑 로그 간의 동작 중에 전원 오류가 발생할 경우 최신 매핑 업데이트의 요청이 소실될 수 있음을 보여준다. 그림 18-(a)에서 논리 주소 A에 쓰기 요청된 호스트 데이터가 데이터 로그 D의 플래시 로그에 기록되면, 상위의 매핑 로그 M0로 해당 논리 주소의 매핑 엔트리에 대한 업데이트(논리 주소와 물리 주소의 쌍)를 요청하는 메시지가 전

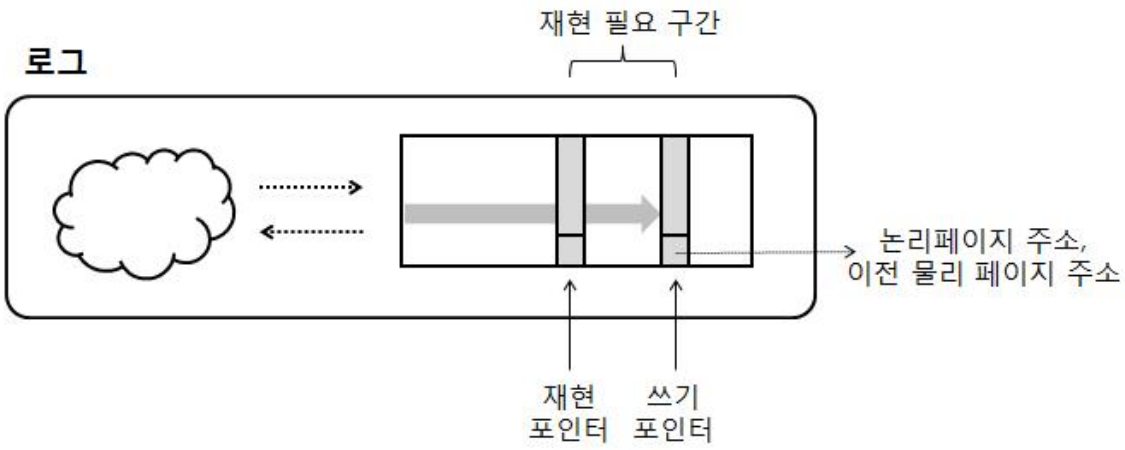

그림 19 기능적 복구를 위한 자료 구조

달된다. 이러한 요청은 곧바로 플래시 메모리에 기록되지 않고 매핑 로그의 캐시에 저장되었다가 추후에 플러시된다. 그림18-(b)는 논리 주소 A에 대한 매핑 업데이트 요청이 매핑 로그의 플래시 메모리에 영속적으로 기록된 것을 나타낸다. 또한 논리 주소 B에 대한 새로운 호스트 쓰기 요청이 들어와 데이터 로그에 영속적으로 저장되고 이에 따라 다시 새로운 매핑 업데이트 요청이 전달되어 캐시에 저장된 상태를 보여준다. 그림 18-(c)는 이 시점에 전원 오류가 발생할 경우 매핑 로그에서 논리 페이지 B에 대한 최신의 매핑 업데이트 요청은 소실되므로 논리 페이지 B에 대한 최신 호스트 데이터를 찾을 수 없게 됨을 보여준다.

이렇게 최신 데이터를 소실하는 문제를 해결하고, 저장 장치 시스템의 논리적인 최신 상태를 복구하기 위해 HIL 프레임워크에서는 상위 로그로의 업데이트 요청을 재현(replay)하는 방법을 사용한다. 그림 19는 이러한 기능적 복구를 위한 자료구조를 보여준다. 기능적 복구를 위해서는 재현이 필요한 정보를 플래시 메모리 페이지의 예비 영역에 기록하는 것이 필요하다. 위에서 예를 든 매핑

관계에서는 페이지의 논리 주소를 저장하는 것이 필요하며, 물리 블록 정보 로그로의 무효화 정보 업데이트 요청을 위해서는 해당 페이지에 의해 무효화되는 기존의 이전 물리 페이지 주소를 기록하는 것이 필요하다.

재현이 필요한 페이지들을 판별하기 위해서 재현 포인터라는 자료 구조를 유지한다. 상위 로그로의 업데이트 요청이 플래시 메모리에 영속적으로 기록되면 상위 로그는 하위 로그에 installed_v 메시지로 응답하고 메시지를 받은 하위 로그는 재현 포인터를 전진시킨다. 즉 재현 포인터 이전의 페이지들은 상위 로그에서 영속화가 되었기 때문에 전원 오류 복구 시에 재현이 필요하지 않다는 것을 나타낸다. 재현 포인터는 쓰기 포인터와 함께 주기적으로 체크포인트 로그에 기록되므로 기능적 복구 단계에서는 재현 포인터 이후의 요청들에 대해서만 재현 작업을 수행하면 된다.

제 4 장　쓰레기 수집 동작의 정확성 검증

플래시 메모리 기반 저장 장치 시스템은 호스트 데이터 뿐 아니라 다양한 FTL 메타데이터의 비휘발성 상태와 휘발성 상태를 유지 및 관리해야 하므로, 비동기적인 전원 오류가 발생했을 때 일관성 있는 저장 장치 상태로 복구하는 것은 매우 어려우면서도 중요한 작업이다. 전통적으로 저장 장치 시스템의 정확성 검증을 위해서는 다양한 워크로드를 이용한 테스트와 시뮬레이션이 사용되어져 왔지만, 이는 매우 오랜 시간이 걸리며 가능한 모든 오류를 찾아낸다는 보장을 할 수 없다. 또한 HIL 프레임워크의 목적은 특정 FTL 구현에 의존적이지 않은 독립적인 전원 오류 복구 방법을 제시하는 것이므로, 하나의 FTL 구현에 대하여 테스트 및 시뮬레이션을 수행하는 것으로는 그 정확성을 검증하였다고 할 수 없다. 이를 극복하기 위해, 최근에는 수학적 모델링에 근거한 정형적 검증(formal verification) 방법을 적용하는 경우가 증가하고 있다. 이는 시스템의 상태 및 동작을 추상적으로 모델링하고 어떠한 경우에도 정확성 기준을 만족한다는 것을 수학적으로 증명하는 방법론이라고 할 수 있다.

쓰레기 수집 작업이 활성화되지 않은 경우, HIL 프레임워크를 통해 구현된 FTL은 어떠한 비동기적인 전원 오류에도 저장 장치 시스템의 일관성을 정확하게 복구한다는 것은 이전 연구[13]를 통해 정형적으로 검증된 바 있다. 본 논문에서는 이를 확장하여 쓰레기 수집 작업이 활성화되는 경우에도 HIL 프레임워크의 정확성이 유지된다는 것을 검증한다.

먼저 쓰레기 수집 작업의 개입에 의해 발생 가능한 문제점에 대해 설명하고, 이러한 문제점을 회피하기 위한 HIL 프레임워크의 로그 운영 규칙을 제시한다.

이를 기반으로 HIL 프레임워크가 쓰레기 수집이 활성화 되는 경우에도 저장 장치 시스템의 정확성 기준을 항상 만족한다는 것을 보인다.

4.1 쓰레기 수집 동작시 전원 오류 복구 문제와 회피 규칙

본 절에서는 HIL 프레임워크를 이용하여 FTL을 구현할 경우, 쓰레기 수집 동작이 활성화 되었을 때 전원 오류에 의해 저장 장치 시스템의 일관성에 문제가 발생할 수 있는 시나리오를 설명하고 이를 회피하기 위하여 추가되어야 하는 규칙을 제시한다.

4.1.1 형제 페이지에 의한 소거 오류 문제

그림 20은 MLC 플래시 메모리에 존재하는 형제 페이지에 의해서 쓰레기 수집 작업과 관련하여 전원 오류 시 발생 가능한 문제점을 보여주고 있다.

그림 20-(a)는 논리 주소 X에 대한 데이터가 물리 페이지 i에 저장되어져 있으며 페이지 i는 물리 블록 B의 유일한 유효 페이지인 것을 나타내고 있다. 그림 20-(b)는 호스트 요청에 의해서 논리 주소 X에 대한 덮어 쓰기 요청이 로그 내부의 캐시를 거쳐 플래시 로그에 기록되는 경우를 보여준다. 데이터가 로그의 최전방의 페이지 j에 기록된 후에 매핑 로그로의 매핑 업데이트 요청(install_mapping)과 물리 블록 정보 로그로의 페이지 i에 대한 무효화 요청(install_invalidation)이 전송된다. 그림 20-(c)에서 물리 블록 정보 로그는 블록 B의 모든 페이지가 무효화된 것을 확인한 후에 해당 블록을 소거하게 된다. 그림 20-(d)는 페이지 j의 형제 페이지에 쓰기 연산이 가해지는 동안 전원 오류가 발생한 상황을 보여준다.

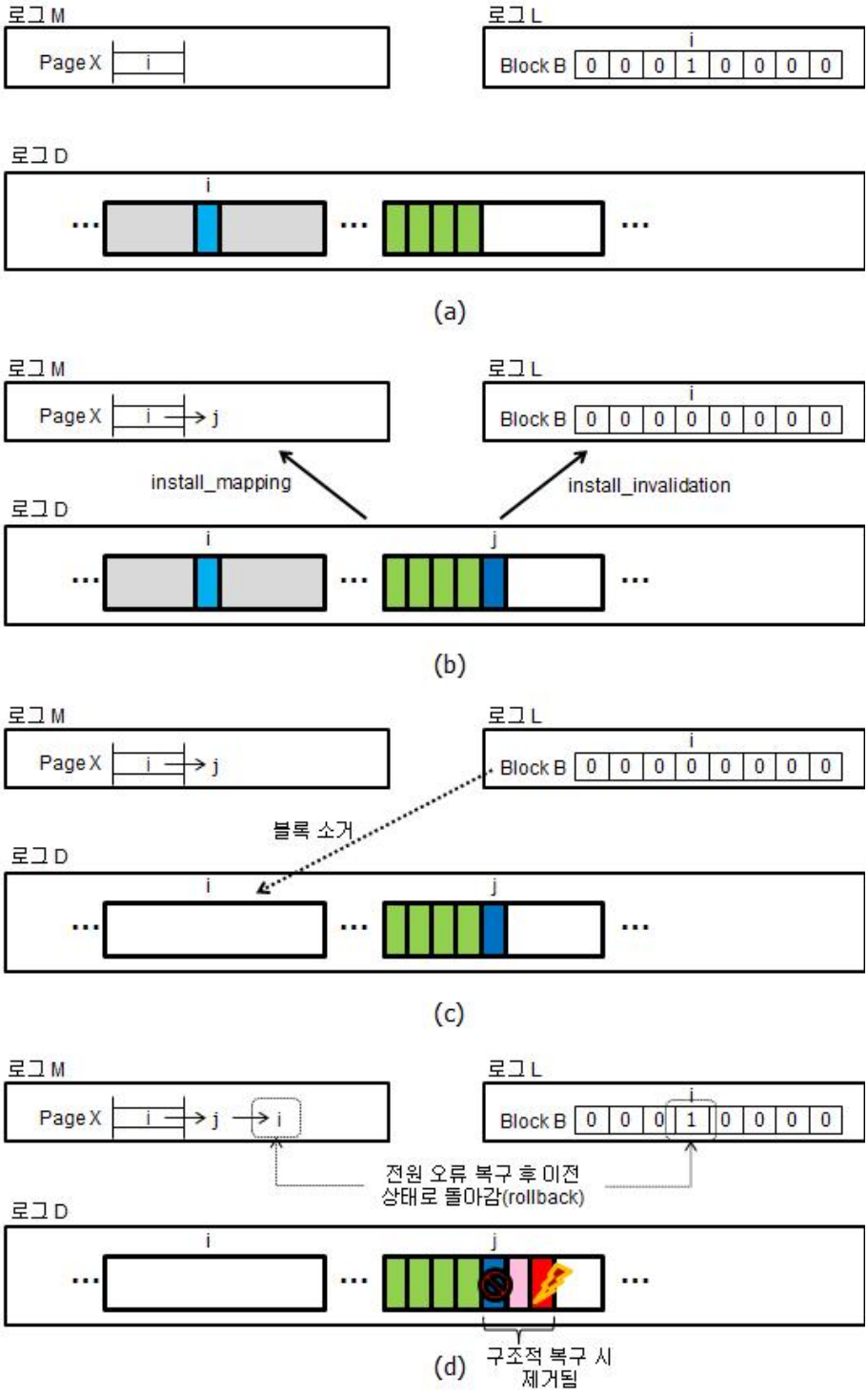

그림 20 형제 페이지에 의한 소거 오류 문제

이 경우, 형제 페이지에 대한 쓰기 동작 중 발생한 전원 오류에 의해 페이지 j에 기록된 데이터 또한 손상을 입게 된다. 그러므로 구조적 복구 작업 도중에 페이지 j는 잔여효과가 있는 페이지로 간주되어 쓰기 가능한 소거 상태로 변경된다. 결국 기능적 복구 작업 중에 페이지 j에 의한 매핑 업데이트 요청이 전송되지 않으므로, 매핑 로그에서 논리 주소 X에 대한 데이터는 물리 페이지 i에 있는 것으로 가리키게 된다. 하지만 이 경우 페이지 i가 속한 블록 B는 이미 소거되어져 있기 때문에 저장 장치 외부에서 볼 때 논리 주소 X에 대한 데이터가 소실되는 현상이 발생한다.

이러한 문제점이 발생하는 이유는 MLC 플래시 메모리 칩으로 로그를 구성할 경우, 페이지에 기록된 데이터의 영속성은 해당 페이지와 쌍을 이루는 형제 페이지에 대한 쓰기가 완료되는 시점으로 미루어지기 때문이다. 그리고 이렇게 영속성이 보장되지 못한 페이지에 의해 발생한 install_invalidation 요청에 의해 물리 블록이 소거될 경우, 구조적 복구 및 기능적 복구 단계를 거치면 매핑이 이미 지워진 예전 물리 페이지를 가리키게 되어 해당 논리 페이지의 내용은 이전 데이터를 잃고 소거된 상태로 호스트에 보여지게 되며, 이는 저장 장치 시스템의 정확성 기준을 어기게 된다. 이 예에서는 데이터 로그에 새로운 페이지가 기록되는 경우를 보이고 있지만 M0, M1 등의 매핑로그에 대한 페이지 쓰기에 대해서도 동일한 문제가 발생한다. 이 경우 호스트 데이터에 손실이 발생하지는 않지만, 매핑 페이지의 소실에 의해 호스트로부터의 요청이 임의의 잘못된 페이지로 보여지는 오류가 발생한다.

이와 같은 문제점을 회피하기 위해 쓰레기 수집과 관련하여 HIL 프레임워크에 아래와 같은 규칙을 추가한다.

[Rule1:deferred_invalidation]

플래시 로그에 기록되는 물리 페이지에 대한 무효화 요청은 해당 페이지에 대한 영속성이 보장된 후까지 지연하여 전송한다.

이와 같은 규칙을 지킬 경우 물리 블록 정보 로그에서 참조하는 페이지의 유효성 정보를 무효화 시킨 모든 페이지는 그 영속성을 보장받을 수 있기 때문에 전원 오류 복구 시에도 일관성에 영향이 없음이 보장된다.

4.1.2 병렬성에 의한 유효 페이지 복사 문제

그림 21은 HIL 프레임워크에서 가정하고 있는 로그 들 간의 병렬성에 의하여 쓰레기 수집 작업 중 유효 페이지 복사 작업 시에 일관성 문제가 발생하는 상황을 보여주고 있다.

그림 21-(a)는 현재 논리 주소 X에 대한 데이터는 물리 페이지 i에 저장되어져 있고 페이지 i는 물리 블록 B의 유일한 유효 페이지인 것을 나타내고 있으며, 플래시 로그의 물리 페이지 j에 새로운 데이터가 쓰여진 것을 나타낸다. 그림 21-(b)는 새롭게 쓰여진 페이지 j에 의해 매핑 로그 및 물리 블록 정보 로그로의 요청이 전송되는 것을 나타낸다. (j의 영속성은 보장된 것을 가정) 그림 21-(c)에서는 FTL 내부의 필요에 의해 희생 블록으로 블록 B를 선정하고 물리 페이지 i에 대한 유효 페이지 복사(valid data copy) 요청을 전송한 상황을 보이고 있다. 이 경우에서 install_invalidation 요청은 아직 도착하기 않은 것을 가정한다. 전달된 유효 페이지 복사 요청에 의해 해당 로그는 페이지 i를 복사하여 로그의 최전방에 추가로 기록하게 된다. 이 경우, 그림 21-(d)에서 보이는 것과

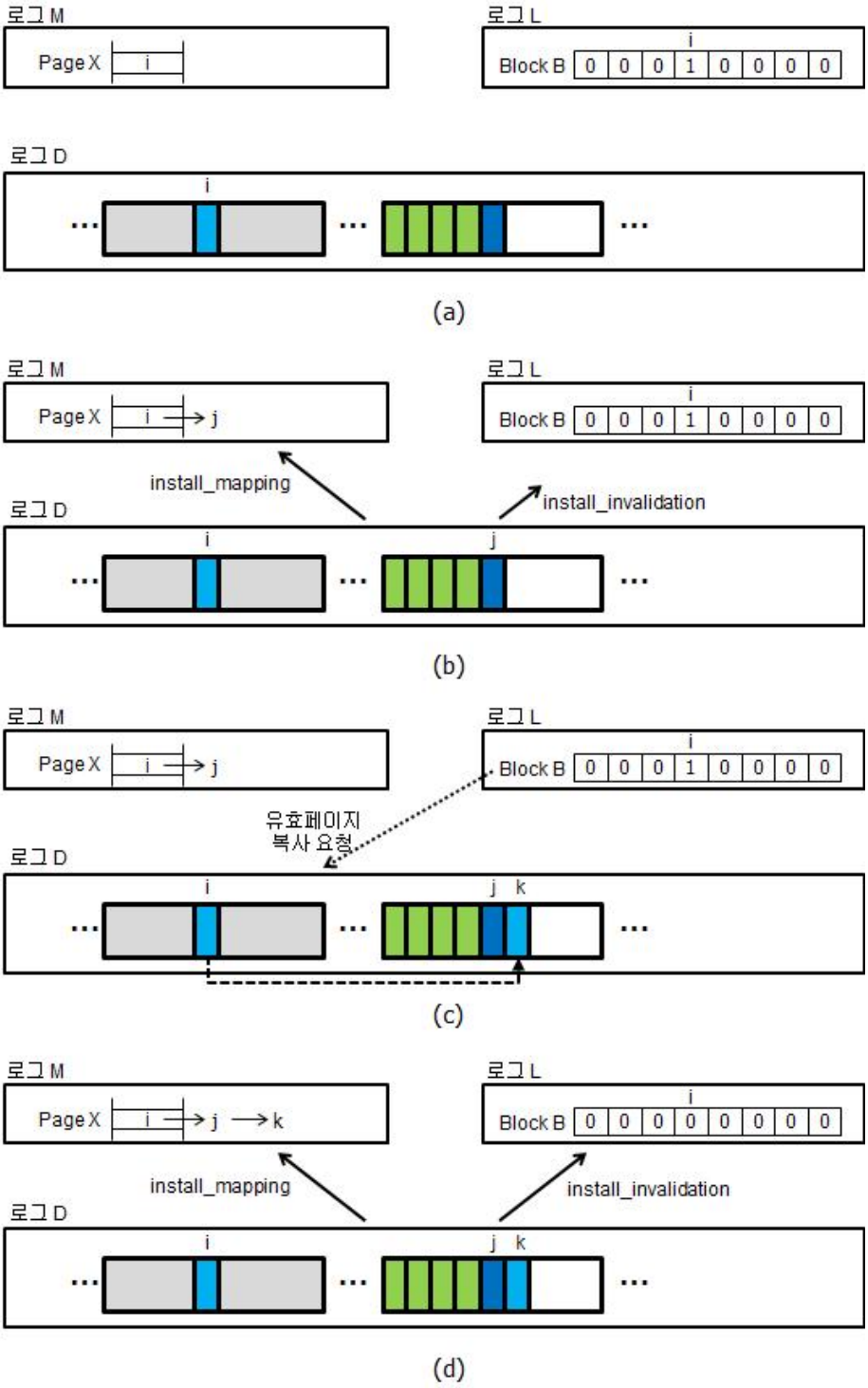

그림 21 병렬성에 의한 유효 페이지 복사 문제

같이, 논리 주소 X에 대한 데이터는 유효 페이지 복사 작업에 의해 새로 기록된 페이지 k를 가리키게 된다. 결국 논리 주소 X는 최신 데이터인 j를 가리키지 않고 예전의 데이터인 k를 가리키게 되므로, 저장 장치의 정확성 기준을 위배하게 된다.

이러한 문제가 발생하는 근본적인 이유는 HIL 프레임워크에서 각각의 로그를 담당하는 쓰레드가 병렬적으로 동작하기 때문이다. 즉, 어느 한 시점에 쓰레기 수집 동작이 발생할 경우 참조하는 물리 블록 정보 로그 내의 유효성 정보는 실제 유효성 정보보다 시간적으로 뒤쳐진 정보라고 할 수 있다. 이렇게 각각의 로그에서 물리 블록 정보 로그로의 유효성 정보 전파가 늦어지는 원인은 1) 로그가 호스트 쓰기 요청을 DRAM에 캐싱을 하고, 2) 형제 페이지의 영속성이 보장될 때 까지 무효화 요청을 지연하며(Rule1:deferred_invalidation), 3) 메시지 전송에 지연 시간이 발생할 수 있기 때문이다. 이러한 환경에서 병렬적으로 동작하는 쓰레드 간에 동기화가 이루어지지 않을 경우 전원 오류 복구 작업 시에 상기한 바와 같은 문제점이 발생할 수 있다.

이러한 병렬성에 의한 무효화 정보의 지연 문제를 회피하기 위하여 HIL 프레임워크에 다음과 같은 규칙을 추가한다.

[Rule2:validity_recheck]

어떤 로그가 유효 페이지 복사 요청을 받은 경우, 해당 물리 페이지를 읽어 예비 영역에 기록된 논리 주소를 확인한 후에, 로그의 캐시에 동일한 논리 주소를 가진 요청이 이미 있을 경우, 또는 예비 영역에서 확인한 논리 주소로 매핑 로그에 질의하여 현재 물리 주소가 복사 요청을 받은 페이지와 다를 경우, 복사 요청을 무시해야 된다.

이와 같은 규칙을 지킬 경우 최신의 무효화 요청이 반영되지 않은 상태에서 물리 블록 정보 로그의 비트맵 정보만을 이용하여 요청된 정확하지 않은 유효 페이지 복사 명령은 수행되지 않게 된다.

4.2 쓰레기 수집 동작 정확성 검증

본 절에서는 HIL 프레임워크에 새롭게 추가된 규칙들을 준수할 경우, 쓰레기 수집 작업이 동작하는 순간 전원 오류가 발생하더라도 구조적 복구와 기능적 복구 단계를 거치면 저장 장치의 정확한 상태로 복원 가능함을 보인다. 먼저 저장 장치 시스템의 정확성 기준에 대해 서술한 후, 쓰레기 수집 작업을 구성하는 세부 작업으로 분류하고, 각각의 경우에 대해서 정확성을 검증한다.

4.2.1 정확성 기준(correctness criteria)

저장 장치 시스템은 기본적으로 사용자 데이터를 비휘발성으로 저장하고 정확하게 추출할 수 있어야 한다. 이러한 관점에서, 정확성 기준은 아래와 같이 정의할 수 있다.

[정확성 기준]

저장 장치 시스템이 주소 공간상의 임의의 논리 페이지 p에 대한 읽기 요청에 대하여 해당 논리 페이지 p에 대한 과거의 쓰기 요청들 중, 가장 최신의 데이터 버전인 v를 항상 반환한다면, 그 저장 장치 시스템은 정확하다고 말할 수 있다.

이는 기존 연구에서 정의하였던 정확성 기준과 동일하며, 쓰레기 수집 동작이 추가 되더라도 이러한 호스트 데이터에 대한 쓰기 및 읽기 작업에 대한 정확성이 유지되어야 한다.

4.2.2 쓰레기 수집 작업의 정확성 검증

쓰레기 수집 작업은 크게 1) 플래시 메모리 저장 장치를 구성하는 모든 물리 페이지의 유효성 정보를 유지하는 작업과 2) 이렇게 유지한 정보를 이용해서 실제로 쓰레기 수집 작업 및 블록 소거 작업을 통해 쓰기 가능한 소거 블록을 만들어 내는 과정으로 나누어서 생각할 수 있다.

물리 페이지의 유효성 정보를 유지하는 작업은 물리 블록 정보 로그(L-log)가 담당하며, 데이터 로그 및 매핑 로그들로부터의 페이지 무효화 요청을 받아 관리 및 저장한다. 물리 블록 정보 로그 역시 관리하는 페이지 유효성 정보를 플래시 메모리에 기록할 수 있는데, 이러한 유효성 정보는 데이터 로그 및 매핑 로그로부터 수동적으로 전달받을 뿐이며, 쓰기 작업 또한 블록 프로그램 리스트에 의해 데이터 로그 및 매핑 로그와 완전히 분리되어 있다. 또한, 호스트 데이터에 대한 읽기 및 쓰기 처리에는 데이터 로그 및 매핑 로그만이 이용된다. 그러므로

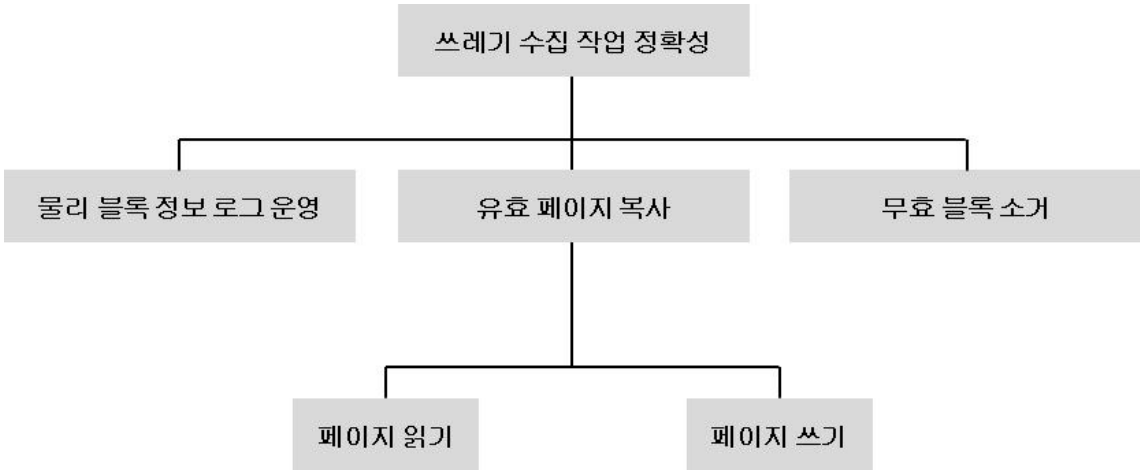

그림 22 쓰레기 수집 작업의 정확성 검증 세부 단계

물리 블록 정보 로그를 운영하는 것 자체로는 전체 플래시 저장 장치의 정확성에는 영향을 끼치지 않는다는 것을 알 수 있다. 쓰레기 수집 동작이 앞에서 정의한 플래시 저장 장치의 정확성에 영향을 끼치는 것은 이러한 정보를 이용하여 실제로 유효 페이지 복사 또는 무효 블록 소거가 발생하는 시점이며, 앞에서 이에 대한 예를 든 바 있다.

사용 가능한 소거 블록의 개수가 FTL이 정한 임계치보다 적어질 경우 쓰레기 수거 동작이 활성화 된다. 만약 모든 페이지가 무효화된 블록이 있다면 해당 블록을 소거하여 즉시 쓰기 가능한 블록을 생산할 수 있다. 모든 페이지가 무효화된 블록이 없을 경우에는 FTL의 정책에 따라 희생 블록(victim block)을 선정한 후에 해당 블록에 남아 있는 유효 페이지를 강제로 복사할 것을 해당 블록을 소유하고 있는 로그에 요청하게 된다.

쓰레기 수집 작업의 정확성은 그림 22와 같이 세부 작업들의 정확성을 검증

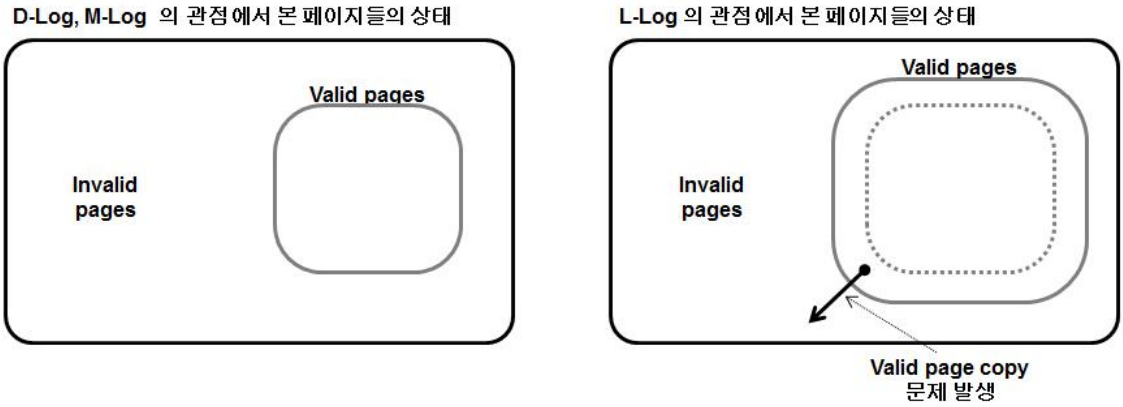

그림 23 로그들 간의 유효성 관점의 차이

하는 것으로 나누어 볼 수 있다.

1) 물리 블록 정보 로그 운영의 정확성

HIL 프레임워크의 기본 설계 방향 중 하나는 모든 로그 들을 비슷한 성질을 가지는 객체로 다룬다는 것이다. 이에 따라 모든 로그들은 구조적 복구 및 기능적 복구를 통해 전원 오류 복구가 가능하고 정확성 검증에 필요한 경우의 수를 줄일 수 있다.

물리 블록 정보 로그도 데이터 로그 및 매핑 로그 등의 일반 로그와 동일하게 캐시와 플래시 로그로 구성된다. 논리 공간에 대한 변경 요청 역시, 페이지 무효화 정보를 받는다는 것과, 여러 로그로부터의 정보 수정 요청을 통합해서 처리한다는 사실을 제외하면 일반 로그와 동일하게 구현되어진다. 그러므로 물리 블록 로그의 논리 공간에 대한 쓰기 요청 시에 전원 오류가 발생하더라도 구조적 복구와 기능적 복구를 통해서 일관된 상태를 복구할 수 있다.

2) 무효 블록 소거 작업의 정확성

그림 23에서 보인 바와 같이, 전원 오류 복구 작업이 끝나고 정상 동작 상태에 들어가면, 모든 로그의 쓰레드가 병렬적으로 동작하더라도, 어느 시점에 L-log에서 참조하는 페이지 유효성 정보에서 모든 페이지가 무효화 되었다는 것은 항상 정확하다는 것이 보장된다. 또한 이러한 블록에 대한 소거 작업은 무해석 블록과 유사하게 블록 내의 내용에 대해 어떠한 가정도 하지 않으므로 잔여 효과에 대한 고려가 필요하지 않다.

소거 작업에 문제가 발생하는 경우는 앞에서 예를 든 것처럼 형제 관계 페이지에 대한 쓰기 작업 중 전원 오류가 발생하는 경우이며, 이는 무효화 정보를 전송하는 것을 지연시키는 규칙을 이용하여 방지하게 된다.

3) 유효 페이지 복사 작업의 정확성

유효 페이지 복사 작업은 해당 페이지를 읽는 작업과 예비 영역에 기록되어 있는 논리 주소를 확인한 후, 로그의 최전방에 기록하는 작업으로 나눌 수 있다.

HIL 프레임워크의 병렬성에 의해 그림 23에서 보인 바와 같이 복사 요청을 전송할 때 무효화된 페이지로 확인된 페이지가 사실은 호스트 요청에 의해 이미 무효화된 페이지일 수 있으며, 이 경우에 단순히 복사 작업을 수행할 경우 호스트가 해당 논리 주소에 쓰기 요청한 최신 데이터가 아니라 이전 데이터의 내용이 보이게 되므로, 앞에서 정의한 저장 장치의 정확성 기준에 어긋나게 된다.

이를 막기 위해 추가된 유효성 재확인 절차를 거치게 되면, 해당 논리 주소에 대한 최신 쓰기 요청이 이미 진행된 경우, 유효 페이지 복사 작업을 수행하지 않

고 무시하게 되므로, 이러한 문제가 발생하지 않음이 보장된다.

복사 작업과 연관된 플래시 메모리 연산을 살펴보면, 페이지 읽기 동작은 플래시 상태에 대한 변경을 발생시키지 않으므로 전원 오류의 영향을 받지 않는다. 유효 페이지 복사는 호스트 쓰기 요청과 달리 논리 주소를 예비 영역에서 가져오고, 내용은 플래시에서 읽는다는 차이가 있을 뿐 플래시 로그에 기록되는 것 자체는 일반 호스트 쓰기 동작과 동일하게 처리 가능하다. 결국 전원 오류에 대해 문제가 발생하지 않는 다는 것을 알 수 있다.

이상과 같이 쓰레기 수집 작업을 구성하는 각각의 세부 단계에 대한 정확성이 보장되므로, 어느 시점에 전원 오류가 발생하더라도 저장 장치 시스템의 정확성에는 문제가 발생하지 않음이 보장된다.

제 5 장　쓰레기 수집 성능 최적화

HIL 프레임워크는 비동기적으로 발생하는 어떠한 전원 오류 상황에서도 성공적으로 저장 장치 시스템의 정확성을 복구할 수 있도록 하는 전원 오류 복구 방법을 제시하는 것을 목표로 한다. 또한 다수의 플래시 메모리 칩과 다수의 플래시 메모리 컨트롤러가 제공하는 병렬성을 최대한 이용하여 저장 장치 성능 향상을 도모한다. 하지만 정확하고 검증 가능한 전원 오류 복구를 보장하기 위하여 추가된 HIL 프레임워크의 로그 운영 규칙들은 FTL 동작의 병렬성을 제한하는 요인으로 작용하여 성능 향상의 걸림돌이 될 수 있다.

본 장에서는 전 장에서 설명한 병렬성에 의한 유효 페이지 복사 문제를 해결하기 위한 유효성 재검사를 보다 효율적으로 할 수 있는 기법을 제시한다. 제안하는 기법은 가상 클록 개념과 블룸 필터를 이용하여 각각의 로그에서 유효 페이지 복사 요청을 받을 경우 불필요한 사상 정보 참조의 횟수를 최소화한다. 먼저 일반적인 블룸 필터의 개념에 대해 설명한 후, 쓰레기 수집 작업에 적용하는 방법에 대하여 설명한다.

5.1 블룸 필터(Bloom Filter)

블룸 필터는 주어진 집합에서 특정 원소의 존재 유무를 판별하기 위한 확률적 자료구조로 고정 크기의 비트벡터와 다수 개의 해시 함수로 구성된다[30]. 유지해야 되는 정보의 양을 최소화하기 위해 집합에 삽입되는 원소들을 모두 저장하는 방식을 사용하지는 않으므로 정확히 어떤 원소들이 있는 지에 대한 정보는 알 수 없지만 해당 원소의 존재 유무만 판별할 수 있다.

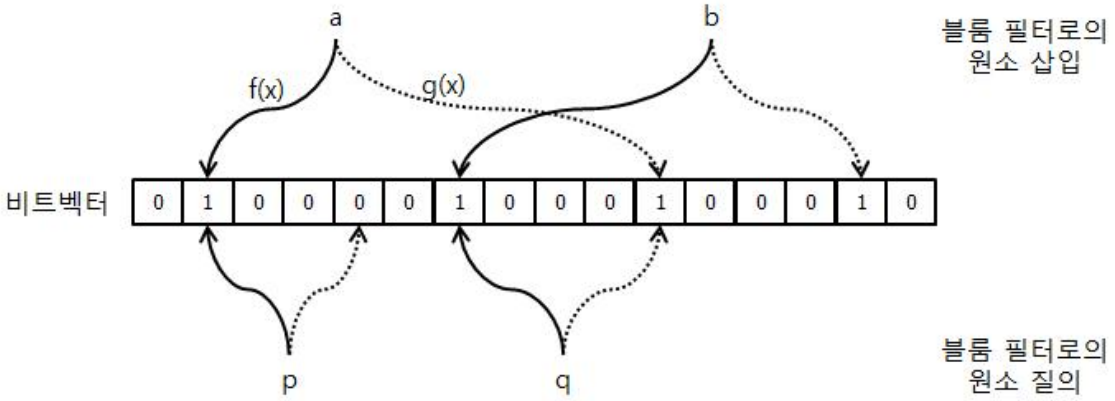

그림 24 블룸 필터로의 원소 삽입과 질의

블룸 필터는 적은 양의 자원으로 빠르게 존재 유무를 판별할 수 있지만 거짓 양성을 허용할 수 있다는 단점이 있다. 즉 블룸 필터에 질의한 결과, 해당 원소가 존재하지 않는 경우에도 존재한다고 응답할 수 있다. 반대로 블룸 필터에 질의한 결과 해당 원소가 존재하지 않는다고 응답했다면 이는 실제로 해당 원소가 집합에 존재하지 않는다는 것이 보장된다. 즉 블룸 필터는 거짓 양성은 발생할 수 있지만 거짓 음성은 발생하지 않는다는 것이 보장된다.

그림 24는 두 개의 해시 함수 f(x), g(x)와 n 비트 길이를 갖는 비트 벡터로 구성된 블룸 필터를 보여준다. 블룸 필터에 어떤 원소를 삽입할 경우, 해시 함수를 통해 얻은 해시 값을 이용하여 해당 위치의 비트 벡터의 값을 1로 설정하게 된다. 그림에서는 원소 a, b 에 의해 비트 벡터의 해당 위치가 1로 설정된 것을 볼 수 있다. 블룸 필터에 어떤 원소의 존재 유무를 질의할 경우에도 해시 함수가 이용된다. 그림에서 원소 p에 대한 질의 시, f(p)는 1이지만, g(p)는 0으로 되어 있으므로, 블룸 필터는 존재하지 않는다고 응답하게 되며(음성), 거짓 음성을 허

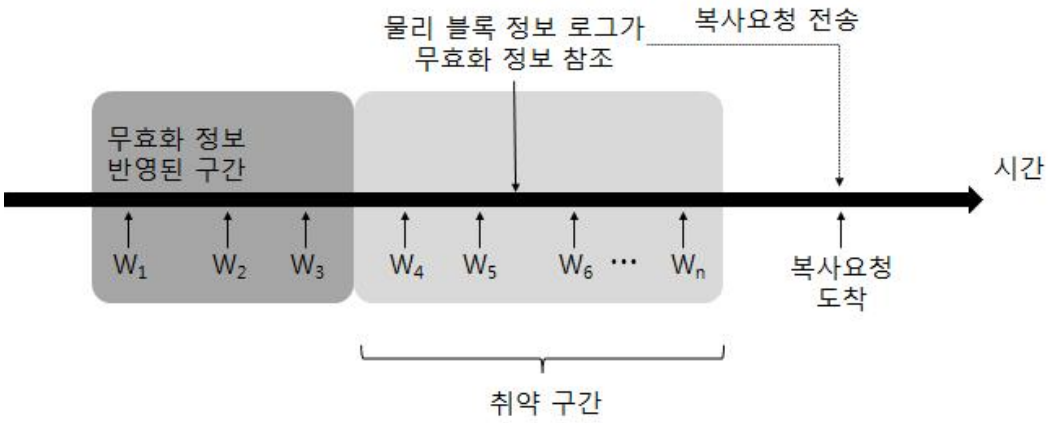

그림 25 시간 관점에서 본 유효 페이지 복사 과정 시 동기화 문제

용하지 않는 블룸 필터의 특성상 실재로도 존재하지 않음이 보장된다. 반면에 원소 q에 대한 질의 시에는, f(q)와 g(q) 두 해시함수의 값이 모두 1이기 때문에 블룸필터는 원소 q가 집합에 존재한다고 응답하게 되지만(양성) 실제로는 존재하지 않는 거짓 양성인 것을 보여주고 있다.

블룸 필터는 거짓 양성이 발생할 수 있다는 단점이 있지만, 적은 양의 자원으로 특정 원소의 존재 유무를 빠르게 판별할 수 있기 때문에 다양한 분야에서 활용되어져 왔다. 다음 절에서는 블룸 필터를 이용하여 쓰레기 수집 작업 시 유효 페이지 성능을 향상시키는 기법을 제시한다.

5.2 가상 클록과 블룸 필터를 이용한 쓰레기 수집 성능 향상

그림 25는 전 장에서 설명하였던 쓰레기 수집 작업 시 유효 페이지 복사 과정에서 발생할 수 있는 동기화 문제를 시간의 관점에서 다시 한 번 살펴본 것으로 W_1, ..., W_n 은 호스트로부터 요청된 각각의 쓰기 요청을 나타내고 있다. 데이

터 로그에서 쓰기 요청된 데이터를 플래시 로그에 기록하고 영속화가 되면 페이지의 유효성 정보를 관리하는 물리 블록 정보 로그로 무효화 요청이 전송 된다. 전 장에서 설명된 것처럼, 이러한 무효화 요청이 전송되어 실제로 페이지 유효성 정보에 반영이 되는데 까지는 시간차가 발생하게 된다. 이런 이유로 쓰레기 수집 작업 시에 참조되는 무효화 정보에는 반영이 안 된 최신 호스트 쓰기 요청이 다수 존재함을 알 수 있다. 만약 무효화 정보 참조 시점에 어느 호스트 쓰기 요청까지 반영이 되었는지에 대한 정보를 전달할 수 있다면 데이터 로그가 유효 페이지 복사 요청을 받았을 때 해당 시점 이후의 요청들과 중첩되는 논리 주소가 있는지만 검사하면 된다. 이를 위해 가상 클록이라는 논리적인 시간의 개념을 도입한다.

가상 클록은 호스트 쓰기 요청과 대응되는 논리적 시간의 단위로서, 요청을 받을 때마다 1씩 증가된다. 쓰기 요청된 데이터가 플래시메모리에 기록되고 영속화된 것이 보장되면, 데이터 로그는 무효화 요청을 가상 클록과 함께 전송한다. 이를 받은 물리 블록 정보 로그는 무효화 요청 처리 후 자신의 가상 클록으로 설정한다. 이렇게 물리 블록 정보 로그가 자신의 가상 클록을 전달 받은 가상 클록 T_v 로 설정하는 것은, T_v 이전의 모든 호스트 쓰기 요청은 데이터 로그에 영속적으로 기록이 되었으며, 이 요청들에 의해 발생한 무효화 요청 역시 물리 블록 정보 로그에 반영이 되었다는 것을 의미한다.

추후에 쓰레기 수집 작업이 시작되어 희생 블록을 선정하고 유효 페이지에 대한 복사 요청을 전달할 때 참조 시점의 가상 클록을 함께 전송하면, 이를 받은 데이터 로그는 가상 클록 이후의 쓰기 요청에 대해서만 중복 검사를 수행하면 된다. 그림 25의 예에서, 쓰레기 수집 작업을 위해 페이지 유효성 정보를 참조

할 시점의 물리 블록 정보 로그의 가상 클록은 W_3으로 설정되어 있을 것이며, 유효 페이지 복사 요청을 전송 시 참조 시점의 가상 클록 W_3을 같이 전송한다. 이 경우, 페이지 복사 요청을 받은 데이터 로그는 W_3 이후의 호스트 요청에 대해서만 논리 주소 중복 여부를 검사하면 된다.

이러한 중복 검사를 위해 모든 호스트 쓰기 요청들을 메모리에 기록하는 것은 비효율적이며 현실적인 방법이라 할 수 없다. 다음에 제시하는 최적화된 기법에서는 공간 효율적이며 빠른 검색이 가능한 블룸필터와 가상클록을 결합한 자료구조를 이용한다.

블룸 필터는 HIL 프레임워크를 구성하는 각각의 로그에서 개별적으로 관리된다. 데이터 로그의 경우 블룸 필터의 가상 클록은 해당 블룸 필터가 초기화 되는 시점의 가상 클록으로 설정되고 호스트 쓰기 요청이 올 때마다 해당 논리 주소가 블룸 필터에 삽입된다. 결국 가상 클록 T_v에 초기화된 블룸 필터는 T_v 이후의 모든 호스트 쓰기 요청을 기록하고 있게 된다.

물리 블록 정보 로그가 복사할 물리 페이지 주소와 무효화 정보 참조 시점의 가상 클록 $T_{v'}$를 전송하면 이를 받은 데이터 로그는 $T_{v'}$ 이후에 존재하는 쓰기 요청의 존재 여부를 블룸 필터에 질의하게 된다. 질의 결과가 음성인 경우, 블룸 필터의 특성상 동일한 논리 주소를 갖는 호스트 쓰기 연산이 없었음이 보장되기 때문에 복사 요청을 수행하면 된다. 반대로 질의 결과가 양성인 경우에는 거짓 양성일 수 있으므로 기존 방법과 같이 상위의 매핑 로그로 질의하여 동일한 논리 주소에 해당하는 호스트 쓰기 연산이 있었는지 확인 하여야 한다.

블룸 필터의 특성 상, 삽입되는 원소의 개수가 증가할수록 거짓 양성을 반환

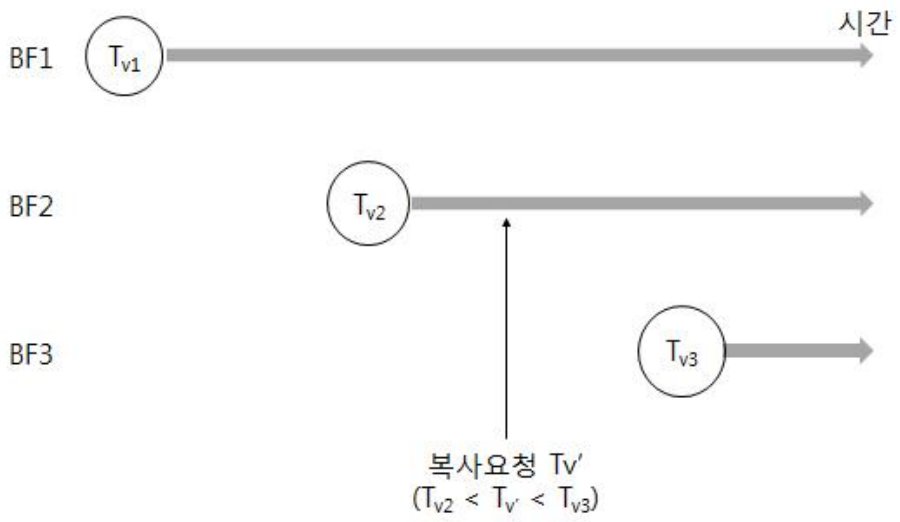

그림 26 다수 개의 블룸 필터를 이용한 방법

할 확률이 증가하기 때문에 이를 감소시키기 위해서는 주기적으로 블룸 필터를 재설정할 필요가 있다. 본 동기화 기법에서는 블룸 필터를 어느 시점에서나 재설정할 수 있지만, 재설정 주기가 너무 짧은 경우 해당 블룸 필터에 질의를 하지 못하게 되는 경우가 자주 발생하게 된다. 이와 반대로 재설정 주기가 너무 길면 거짓 양성을 반환할 확률이 높아지기 때문에 적당한 재설정 주기를 설정해 주는 것이 중요하다.

다수 개의 블룸 필터를 시간차를 두고 중첩하여 사용하면 블룸 필터가 거짓 양성을 반환하는 확률을 더욱 줄일 수 있다. 그림 26은 3개의 블룸 필터를 시간차를 두고 초기화하여 사용하는 예로, BF1, BF2, BF3 블룸 필터는 각각 T_{v1}, T_{v2}, T_{v3} 이후의 호스트 쓰기 요청을 기록하고 있다. 이 경우, 복사 요청이 가상 클록 $T_{v'}$($T_{v2} < T_{v'} < T_{v3}$)와 함께 전송 된다면 BF2 블룸 필터에 질의하면 거짓 양성 반환 확률을 최소화할 수 있다. BF3 블룸 필터는 $T_{v'}$~T_{v3}의 호스트 쓰기

요청을 기록하지 못하기 때문에 사용할 수 없다. T_{v1} 이후의 호스트 쓰기 요청을 기록하고 있는 BF1 블룸 필터에 질의하여도 정확성에는 문제가 없지만 T_{v2} 이전의 호스트 쓰기 연산 또한 기록하고 있으므로 BF2 블룸 필터에 질의한 경우보다 거짓 양성을 반환할 확률이 높다.

따라서 가상 클록과 블룸 필터를 결합한 동기화 기법을 사용하면 유효 페이지 복사 작업 시 상위 로그로의 불필요한 질의를 최소화하여 기본적인 방법만을 적용하였을 경우의 성능 저하 현상을 개선할 수 있다.

제 6 장　결론 및 향후 연구 과제

컴퓨터 시스템의 주된 용도 중 하나인 데이터를 안정적으로 저장하고 효율적으로 추출하는 작업을 담당하는 저장 장치 시스템은 운영체제의 파일시스템으로부터 저장 장치 내부의 펌웨어까지 매우 복잡한 소프트웨어로 구성되어 진다. 저장 장치 시스템 연구의 중요한 두 축은 성능 및 신뢰성이며 전통적인 하드디스크의 특성에 최적화된 많은 연구가 수행되어져 왔다.

플래시 메모리는 초기에는 내장형 시스템이나 이동형 장치에 주로 사용되었지만, 최근에는 고성능 서버 시스템의 저장 장치 매체로까지 급속하게 적용 범위가 넓어지고 있으며, 그에 따라 소프트웨어의 복잡도 역시 급격하게 증가하고 있다. 플래시 메모리 기반 저장 장치 시스템에 대한 연구는 덮어쓰기가 불가능한 플래시 메모리의 제약 조건을 극복하기 위한 매핑 알고리즘 관점에서 수행되어져 왔으며, 신뢰성 관련 연구는 상대적으로 많은 연구가 이루어지지 않았다. 최근에는 집적도 증가를 위한 미세 공정 도입 및, MLC 타입 플래시 메모리의 등장으로 플래시 메모리의 신뢰성 특성은 더욱 악화되고 있으며, 비동기적인 전원 오류에 대해 정확한 복구 작업을 하는 것은 점점 더 어려운 작업이 되고 있다.

HIL은 체계적이고 검증 가능한 전원 오류 복구를 목표로 하는 FTL 설계 프레임워크로, 특정 매핑 알고리즘에 국한되지 않는 다양한 FTL을 설계 및 구현할 수 있는 방법을 제공한다. HIL 프레임워크는 데이터의 영속성을 제공하는 논리 공간을 표현하는 로그와 사용자 데이터 및 FTL 메타 데이터 등 다양한 로그 간의 계층적 구성을 통해 FTL을 구현한다. 또한 체계적인 전원 오류 복구를 위

해 로그 운영 및 로그 간의 상호 작용시에 필요한 규칙들을 명시하여, 특정 매핑 알고리즘에 국한되지 않고 다양한 FTL에 적용 가능한 전원 오류 복구를 가능케 한다.

본 논문은 HIL 프레임워크에 쓰레기 수집 작업을 위한 물리 블록 정보 로그를 정의하고 사용자 데이터 및 매핑 관련 FTL 메타데이터를 저장하기 위한 로그들과의 전원 오류 복구를 위한 인터페이스 및 로그 운영 규칙을 제시하였다. 물리 블록 정보 로그는 플래시 메모리 저장 장치 시스템 내의 모든 물리 페이지에 대한 유효성 정보를 유지 및 관리하며, 쓰기 가능한 미리 소거된 자유 블록의 개수가 부족해질 경우 쓰레기 수집 작업을 수행하게 된다. 전원 오류 발생시, MLC 타입 플래시 메모리의 형제 페이지 관계에 의해 소거 오류 문제가 생길 수 있으며, 각각의 로그가 병렬적으로 동작할 경우 쓰레기 수집 작업 중 유효 페이지 복사 시 문제가 발생할 수 있음을 보이고, 이러한 문제점 들을 회피하여 정확한 전원 오류 복구를 가능케 하는 로그 운영 규칙을 추가하였다. 이를 통해 HIL 프레임워크를 이용해 설계된 FTL이 비동기적인 전원 오류 발생 시에도 정확한 저장 장치 상태로 복구될 수 있음을 검증하였다. 또한 유효 페이지 복사 작업 시 잦은 논리 주소 중복 검사로 인해 발생할 수 있는 성능 저하를 최소화하기 위하여 가상 클록 및 블룸 필터를 활용하는 기법을 제시하였다.

본 논문은 쓰레기 수집 작업과 관련된 HIL 프레임워크의 전원 오류 복구 작업에 대한 이론적인 정확성 검증에 초점을 맞추어 서술되었다. 향후 연구로는 실제로 HIL 프레임워크에 의해 구현된 다양한 플래시 저장 장치 시스템에서 실험을 통해 제안된 전원 오류 복구의 성능 오버헤드를 측정하는 것이 의미가 있을 것이다. 이를 위해서는 저장 장치의 외부에서 임의의 시점에 전원 오류를 발

생시키거나, FTL 펌웨어에서 직접 전원 오류를 주입하는 등 전통적으로 저장 장치 시스템 연구에서 이용되어져 온 다양한 방법을 사용할 수 있을 것이다. 플래시 메모리 저장 장치 시스템의 적용 및 중요성이 더욱 커지고 있는 최근 추세를 볼 때, 정확한 전원 오류 복구를 통한 신뢰성 향상 및 성능에 미치는 영향을 측정하여 최소화하는 것은 의미 있는 작업이 될 것이다.

참고문헌

[1] McKusick, M. K., Joy, W. J., Leffler, S. J., Fabry, R. S., “A Fast File System for Unix,” ACM Trans. on Computer Systems, V 2 (3), pp. 181-197, 1984

[2] M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” ACM Transactions on Computer Systems, vol. 10, no. 1, pp. 26-52, 1992.

[3] V. Prabhakaran, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems," Proceedings of USENIX Annual Technical Conference, 2005.

[4] J. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, “A space-efficient flash translation layer for compactflash systems,” IEEE Transactions on Consumer Electronics, vol. 48, no. 2, pp. 366-375, 2002.

[5] S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park and H. J. Song, “A log buffer-based flash translation layer using fully-associative sector translation,” ACM Transactions on Embedded Computing Systems, vol. 6, no. 3, 2007.

[6] J. U. Kang, H. Jo, J. S. Kim and J. Lee, “A superblock-based flash translation layer for NAND flash memory,” Proceedings of International Conference on Embedded Software, pp. 161-170, 2006.

[7] A. Gupta, Y. Kim and B. Urgaonkar, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 229-240, 2009.

[8] H. Kim and S. Ahn, “BPLRU: a buffer management scheme for improving random writes in flash storage,” Proceedings of USENIX Conference on File and Storage Technologies, pp. 239-252, 2008

[9] Y. H. Chang and T. W. Kuo, "A reliable MTD design for MLC flash-memory storage systems," Proceedings of International Conference on Embedded Software, pp. 179-188, 2010.

[10] Micron Technology Inc, “Bad block management in NAND flash memory,” Technical Note-29-59, 2011

[11] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Computing Surveys, vol. 37, no. 2, pp. 138-163, 2005.

[12] L. M. Grupp, J. Davis and S. Swanson, "The bleak future of NAND flash memory," Proceedings of USENIX Conference on File and Storage Technologies, 2012

[13] Y. J. Seong, "Formal verification of a compositional FTL design framework", doctoral dissertation, Seoul National University, 2013

[14] J. Yun, “X-BMS: a provably-correct bad block management scheme for flash memory based storage systems,” Ph. D. dissertation at Seoul National University, 2010.

[15] E. H. Nam, B. S. Kim, H. Eom and S. L. Min, "Ozone (O3): an out-of-order flash memory controller architecture," IEEE Transactions on Computers, vol. 60, pp. 653-666, 2011.

[16] Y. H. Chang, J. W. Hsieh and T.-W. Kuo, "Improving flash wear-leveling by proactively moving static data," IEEE Transactions on Computers, vol. 59, no. 1, pp. 53-65, 2010.

[17] D. Jung, Y. H. Chae, H. Jo, J. S. Kim and J. Lee, "A group-based

wear-leveling algorithm for large-capacity flash memory storage systems," Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp. 160-164, 2007.

[18] A. Ban, FLASH FILE SYSTEM, US Patent no. 5,404,485, 1995.

[19] S. Lee, D. Shin, Y. J. Kim and J. Kim, “LAST: locality-aware sector translation for NAND flash memory-based storage systems,” ACM SIGOPS Operating Systems Review, vol. 42, no. 6, pp. 36-42, 2008.

[20] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho and J. S. Kim, “A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications,” ACM Transactions on Embedded Computing Systems, vol. 7, no. 4, 2008.

[21] J. H. Yoon, E. H. Nam, Y. J. Seong, H. Kim, B. S. Kim, S. L. Min and Y. Cho, “Chameleon: a high performance flash/FRAM hybrid solid state disk architecture,” IEEE Computer Architecture Letters, vol. 7, pp. 17-20, 2008.

[22] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J. Y. Choi, S. Lee, Y. H. Bae, J. Lee, Y. Cho and S. L. Min, "Hydra: a block-mapped parallel flash memory solid-state disk architecture," IEEE Transactions on Computers, vol. 59, pp. 905-921, 2010.

[23] A. Kawaguchi, S. Nishioka and H. Motoda, “A flash-memory based file system,” Proceedings of USENIX Annual Technical Conference, 1995

[24] S. Park, J. H. Yu and S. Y. Ohm, "Atomic write FTL for robust flash file system," Proceedings of IEEE International Symposium on Consumer Electronics, pp. 155-160, 2005.

[25] J. H. Yun, J. H. Yoon, E. H. Nam and S. L. Min, "An abstract fault model for

NAND flash memory," IEEE Embedded Systems Letters, vol. 4, pp. 86-89, 2012.

[26] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz, "ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," ACM Transaction on Database System, vol. 17, pp. 94-162, 1992.

[27] E. A. Emerson, "The beginning of model checking: a personal perspective," in 25 years of model checking, pp. 27-45, 2008.

[28] Intel Corporation and Seagate Technology, “Serial ATA Native Command Queuing,” White paper, 2003.

[29] M. Wu and W. Zwaenepoel "eNVy: A Non-Volatile, main memory storage system", Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, pp.86 -97, 1994

[30] B.H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

Abstract

Correctness Verification of Garbage Collection Based on a Compositional FTL Design Framework

Sookwan Lee

Department of Electrical Engineering and Computer Science

Graduate School

Seoul National University

Flash memory has been widely accepted as a storage medium not only for embedded and mobile systems but also for high performance server systems. The performance of flash-based storage systems has increased dramatically thanks to extensive research not only on their architecture but also on the constituent parts. However, there has not been much research on improving the reliability of flash-based storage systems. Research on reliability is becoming more and more important because the reliability characteristics of flash memory has been deteriorated by the shrinking geometries of flash memory cells and the use of the MLC(multi-level cell) technology.

Flash-based storage systems are usually of high complexity since they have to manage both volatile and nonvolatile states of not only user data but also metadata of their own. This inherent complexity of flash-based storage systems makes it difficult to restore the integrity of storage system from an asynchronous

power failure.

This research is based on the HIL(Hierarchically Interacting set of Logs) which is a compositional FTL design framework. The basic components of the HIL framework are logs that abstract the characteristics of flash memory. In the HIL, both host data and the FTL's metadata are managed through the interaction of logs.

In this dissertation, the correctness of the crash recovery of the FTL is verified based on the HIL framework. A particular attention is given to the garbage collection part of the FTL. For this purpose, we define a new log type called the physical block information log and give the interface and the management rules for its interaction with user data logs and the FTL's metadata logs. With this setting, we show that FTLs designed using this extended HIL framework can be recovered correctly from any combination of asynchronous power failures. Also, to minimize the performance degradation resulting from unnecessary mapping table look-ups during garbage collection, we propose a new technique based on virtual clocks and bloom filters.

Key words: crash recovery, flash memory, flash memory based storage system, flash translation layer (FTL), garbage collection

Student Number: 2002-21581